Estimation of Chaos Function for the Implementation of High-Resolution Measurement System

Dhrubajyoti Dutta

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Doctor of Philosophy

May 2018

ABSTRACT

The use of chaotic maps as measurement system or as a signal quantisation unit in analogue to digital converters (ADC) is a fairly modern approach. Compared to existing ADC architectures, chaotic systems are advantageous because these are simple mathematical functions and can be implemented physically involving less components. Additionally, unique symbolic identities corresponding to an input value (initial condition) can be generated iteratively through the dynamics, thus simplifying the design complexities.

For the application of signal measurement system, the chaotic function tent map (TM) is found to be the suitable candidate, as dense distribution of points within the state-space can be realised from the dynamics. However, a significant issue that may arise while implementing the TM electronically is that, it is difficult to maintain the parameter of the map at the ideal value due to component imprecisions. When the map parameter is reduced, the dynamics is distorted from the ideal behaviour; hence estimating the initial condition from the symbols become difficult. If the knowledge of the non-ideal parameter is available, then the actual initial condition can be recovered. Hence, it becomes essential to determine the non-ideal parameter from the available dynamics.

In this work, two different approaches have been proposed for the parameter estimation of the TM. The first approach is realised from the symbolic dynamics of the TM in which the sequence corresponding to the map maximum is searched over a symbolic time series, and a difference equation is realised in terms of the map parameter. The second method is based on the identification of the map fixed-point through the noisy dynamics of the TM. It has been discovered that unique crossovers appear within the noisy samples and the information of the map fixed-point is preserved through those crossovers. The proposed methods have been broadly analysed through mathematical simulations and graphical results. The approaches deliver parameter estimates with errors below 1% and using short length trajectories as low as 200 iterations. This development can benefit accurate estimation of initial condition from the non-ideal dynamics and therefore may be considered as a step forward in the development of chaos-based measurement systems and chaotic ADCs. The study and the proposed estimation methods can also be utilised in other areas of applications such as communication and encryption, where parameter estimation of the chaotic functions is one of the prime requirements.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor Dr. Peter Mather whose constant guidance and support, which was not always limited to academics, has been priceless. His patience and encouragement have been indispensable throughout the research as well as its documentation. I would also like to thank Dr. Violeta Holmes for her support and advice throughout my time at the university. Alongside my supervisors, Prof. Soumitro Banerjee of IISER, Kolkata, has been an invaluable source of guidance and for that, I would like to express my deepest gratitude and appreciation.

I have been lucky enough to work with some amazing people, including Dennis Town, whose friendship has often extended beyond the call of duty, and I feel truly blessed to have known him. Also, I am thankful to David Bray and Martin Webster for being indispensable whenever I was toiling in the labs. I would also like to thank Chris Sentance and everyone else at the Research Office for their constant support during my PhD.

I would also like to thank my friends and colleagues who kept the spirits up during this journey. In particular, I thank Rajlaxmi Basu, for the many hours spent over crucial yet interesting discussions. I also thank David Upton, Jan Berkenbusch and Michael Agolom for being the wonderful friends and colleagues that they are. I thank Debkumar Basu for the banter, for being an amazing friend in such a short time and for keeping me sane, especially over the last couple of months. Finally, I would like to thank my parents, Tapan and Swapna for keeping up with me and for their relentless support throughout, but not limited to, the PhD.

TABLE OF CONTENTS

Abstract 1

Acknowledgement 2

List of figures 7

List of tables 10

List of Publications 11

List of abbreviations and acronyms 12

1 Introduction 13

1.1 Chaotic Measurement System: An Overview 16

1.2 Aims & Objectives 20

1.2.1 Realising the Dynamics of TM Over Various Parameters 20

1.2.2 Realising Symbolic Correspondence with the TM Parameter 21

1.2.3 Studying the Behaviour of the TM Fixed Point 21

1.2.4 Investigating the Noisy Dynamics of TM 21

1.2.5 Determining the Methods to Estimate the Parameter 22

1.3 Original Contributions 22

1.4 Organisation 23

2 Background Literature Review 25

2.1 Measurement Theory 25

2.1.1 Accuracy, Precision and Resolution 26

2.2 Analogue to Digital Conversion 28

2.3 Chaotic Dynamics and Definitions 32

2.4 Chaotic Maps 35
2.4.1 Logistic Map 36
2.4.2 Bitshift Map 38
2.4.3 Tent Map 40
2.5 Properties of Tent Map 42
2.5.1 Maximum and Minimum of TM Trajectories 45
2.5.2 Symbolic Dynamics of TM 49
2.5.3 Symbolic Representation of the State Space 50
2.6 Symbolic Shifting Window 56
2.7 Kneading Theory: Symbolic Maximum and Minimum 58
3 Chaos based signal measurement 61
3.1 Basic Challenges 61
3.1.1 Parametric Reduction of the TM 62
3.1.2 Impact on Initial Condition Estimation 64
3.1.3 Dynamical Behaviour Affected by Noise 67
3.2 Parameter Estimation: Current Techniques & Limitations 74
4 Parameter Estimation Methods 78
4.1 Parameter Estimation: Kneading Sequence Approach 78
4.1.1 Proposed Kneading Sequence Search Algorithm 79
4.2 Parameter Estimation from Noisy Dynamics of TM 83
4.2.1 The Algorithm: Parameter Estimation from Noisy Trajectories 87

5 Results: Parameter Estimation 91

5.1 Results: Kneading Sequence Search Algorithm 92

5.2 Results: Parameter Estimation from Noisy Dynamics 100

6 Conclusion and Future Scopes 113

6.1 Future scopes 115

6.1.1 Chaotic Measurement System Implementation 116

6.1.2 Applications of Chaos in Encryption 119

References 121

Appendix 1: Publications 129

Appendix 1.1 130

Appendix 1.2 131

Appendix 2: MATLAB Codes 132

Appendix 2.1: Logistic Map (LM) 133

Appendix 2.2: LM Bifurcation Diagram 135

Appendix 2.3: Bitshift Map (BM) 136

Appendix 2.4: BM Bifurcation Diagram 138

Appendix 2.5: Tent Map (TM) 140

Appendix 2.6: TM Bifurcation Diagram 142

Appendix 2.7: TM Cobweb Diagrams 144

Appendix 2.8: GON for TM 147

Appendix 2.9: TM Shifting Window 150

Appendix 2.10: Kneading Sequence Method (single input)..... 154

Appendix 2.11: Kneading Sequence Method (full dataset)..... 158

Appendix 2.12: Crossover Method (single input)..... 163

Appendix 2.13: Crossover Method (full dataset)..... 167

LIST OF FIGURES

Fig. 1.1 Analogue signal detection using ADC 14

Fig. 1.2 Fundamental block diagram of chaotic ADC 17

Fig. 2.1 Reducing the step size improves resolution 29

Fig. 2.2 Logistic Map 36

Fig. 2.3 Bifurcation diagram of Logistic Map 37

Fig. 2.4 Bitshift Map 39

Fig. 2.5 Bifurcation diagram of Bitshift Map 39

Fig. 2.6 Tent Map 40

Fig. 2.7 Bifurcation diagram of Tent Map 41

Fig. 2.8 Fixed point of the ideal TM 43

Fig. 2.9 xf moves along the map diagonal as the parameter is altered 44

Fig. 2.10 Change of xf with respect to the parameter 45

Fig. 2.11 Cobweb diagram for μ = 0.8; $x0$ = 0.000124 46

Fig. 2.12 Cobweb diagram for μ = 0.8; $x0$ = 0.823 47

Fig. 2.13 Maximum and minimum of a trajectory with μ = 0.8 47

Fig. 2.14 The dynamical attractor I' for a parameter μ < 1 48

Fig. 2.15 Maximum and minimum over parameter 49

Fig. 2.16 Symbolic correspondence to the intervals of the state space 51

Fig. 2.17 Fractal structure of symbolic codes 54

Fig. 2.18 Shifting window mechanism and obtaining GON 57

Fig. 2.19 Trajectories recreated through GON of shifting window 58

Fig. 3.1 Altered dynamics due to reduction in parameter value 63

Fig. 3.2 Partitions get shifted generating asymmetrical intervals 63

Fig. 3.3 GON estimation with reduced parameter 65

Fig. 3.4 Noise affected TM iterations in a feedback mode 68

Fig. 3.5 Dynamics of TM affected by noise in every stage of iteration 69

Fig. 3.6 Divergent noisy trajectories of an initial condition 70

Fig. 3.7 Noisy trajectories with SNR = 30 dB..... 71

Fig. 3.8 Bifurcation diagrams of noise-free and noise affected tent map 72

Fig. 4.1 Operation of shifting window and determining GON of each shift 80

Fig. 4.2 Crossovers around x_f = 0.6226 for μ = 0.825 84

Fig. 4.3 Crossovers around xf = 0.5745 for μ = 0.625..... 85

Fig. 4.4 Mapping within the state space..... 86

Fig. 4.5 Selection of the iterates to determine the intersections..... 88

Fig. 4.6 Assignment of coordinates to the selected samples 89

Fig. 5.1 The equation build-up: differences added to GON($\mathcal{K}$)..... 95

Fig. 5.2 Estimated parameter values for different parametric conditions 96

Fig. 5.3 Percentage error in parameter estimation..... 97

Fig. 5.4 Estimated parameter for all initial conditions 98

Fig. 5.5 Relationship between estimation accuracy and window size 100

Fig. 5.6 Distribution of Yksolutions between n = 16 and 17 101

Fig. 5.7 Distribution of Yk solutions between n = 20 and 21 102

Fig. 5.8 The mean Yn of crossover points (black-square legend) 103

Fig. 5.9 Fixed point crossover estimates for SNR 30dB..... 104

Fig. 5.10 Estimated parameter error bar plot for μ = 0.95..... 106

Fig. 5.11 Estimated parameter error bar plot for μ = 0.90..... 106

Fig. 5.12 Estimated parameter error bar plot for μ = 0.85..... 107

Fig. 5.13 Estimated parameter error bar plot for μ = 0.80..... 107

Fig. 5.14 Estimated parameter error bar plot for μ = 0.75 108

Fig. 5.15 Estimated parameter for all inputs (SNR = 20 dB) 109

Fig. 5.16 Estimated parameter for all inputs (SNR = 25 dB) 110

Fig. 5.17 Estimated parameter for all inputs (SNR = 30 dB) 110

Fig. 6.1 Functional block diagram of the measurement system [24] 117

LIST OF TABLES

Table 2.1 Correspondence between intervals and symbolic sequence 52

Table 3.1 Sequences generated with $\mu = 1$and $\mu < 1$ 64

Table 5.1 Shifting window of 12-bit operated over symbolic sequence 93

Table 5.2 Obtaining Kneading sequence $\mathcal{K}$ from S_{max} 93

Table 5.3 Kneading sequence 94

LIST OF PUBLICATIONS

1. **Parameter Estimation for 1D PWL Chaotic Maps Using Noisy Dynamics**

Reference
Dutta, D., Basu, R., Banerjee, S., Holmes, V., & Mather, P. (2018). Parameter estimation for 1D PWL chaotic maps using noisy dynamics. *Nonlinear Dynamics, 94*(4), 2979-2993.

Personal contributions

- The effect of dynamical noise on discrete chaotic timeseries
- Immergence of crossovers within a collection of timeseries when linear constructions were applied between consecutive iterates
- Correspondence between the crossovers and the chaotic map fixed point
- Statistical estimation of the crossover concentration from the collection of sampled trajectories - to estimate the fixed point and map parameter

2. **An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map**

Reference
Basu, R., Dutta, D., Banerjee, S., Holmes, V., & Mather, P. (2018). An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *65*(7), 2221-2231.

Personal contributions

- Realising the behaviour of map maximum and minimum (dynamical attractor) under varied parametric conditions
- Correspondence between the Kneading sequence and symbolic codes of map maximum and minimum
- Application of the Kneading theory to determine the map parameter

LIST OF ABBREVIATIONS AND ACRONYMS

ΔΣ	Delta Sigma
1D	One dimensional
2D	Two dimensional
3D	Three dimensional
ADC	Analogue to digital converter
AWGN	Additive white Gaussian noise
BM	Bitshift (Bernoulli) map
DAC	Digital to analogue converter
GON	Gray ordering number
LM	Logistic map
LSB	Least significant bit
MSB	Most significant bit
PWL	Piecewise linear
S/H	Sample and hold
SAR	Successive approximation register
SNR	Signal to noise ratio
TM	Tent map
Vdac	DAC output voltage
Vin	Input voltage
Vref	Reference voltage
XOR	Exclusive OR

1 INTRODUCTION

Measurement play a vital role in scientific disciplines to acquire better understanding of the behaviour of nature and engineering systems. Collecting observations from scientific experiments, monitoring and exercising control over engineering applications, involve exhaustive measurement of some physical quantity. Due to the growing needs of performance, it is a prerequisite for the measurement systems to be capable of extracting information with greater levels of accuracy. An efficient measurement system involves several precision components and techniques that deliver the desired degree of accuracy.

The most salient stages of instrumentation that a standard measurement system comprises are the sensing elements (sensors), signal conditioning stage and analogue to digital converters (ADC) [1]. Sensors are typically used to detect changes in physical quantities in the form of electrical signals, and range over different types, catering to a number of different applications. To improve the quality of the sensed signal, conditioning elements e.g. amplifiers and filters are included in the intermediary stage between the sensor and ADC.

Data acquisition and instrumentation systems rely upon good quality ADCs to digitise the sensor signal so that the measured information can be stored and processed in the computation domain [1]. The ADC therefore, is regarded as a significant component in a measurement system. Several essential stages are involved in signal digitisation (illustrated in Fig. 1.1) that include detecting (sampling) an input voltage signal and comparing it with a fixed known reference which is often referred to as quantisation; the compared signal is then assigned a digital value (binary: 1 or 0) depending on whether the input signal has crossed

certain reference threshold or not. This results in the successful conversion of an analogue signal to its digital equivalent [2].

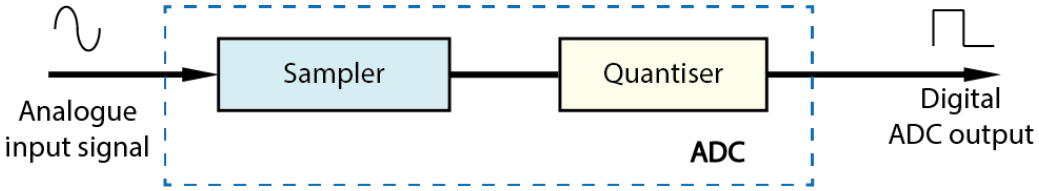


Fig. 1.1 Analogue signal detection using ADC

There are a wide range of ADCs available with different architectures that depend on the performance need and the kind of application for which it is to be dedicated. Flash [3], delta-sigma (ΔΣ) [4], successive approximation (SAR) [5], pipelined [6], and modified flash [7] type ADCs are primarily the widely popular architectures. None of these ADCs, however, completely outmatch the other, since each type of architecture shows certain advantages over the other. As reported by Walden et al. [8] there are many criteria relating to the accuracy, speed, hardware complexity and power consumption that are applied to benchmark the performance of ADCs. Bashir et al. have later summarised in [9], that, there are certain trade-offs between performance, resource consumption and cost that need to be considered while optimally choosing ADCs for different applications. The complexity level of the ADC hardware varies from type to type of the architectures chosen. As a result, when the ADC technologies are further upgraded for better accuracy and resolution these trade-offs often come into play, and therefore optimisation of performance to material cost, enhancement of resource and power consumption is an ongoing process of development.

To optimise the accuracy and design factors of a measurement system, new techniques are investigated. Application of chaotic functions as measurement systems is one such new technique that was first proposed by Michael Peter

Kennedy in [10], where he established the significance of considering chaotic functions as suitable quantisation units. The idea was broadly justified by the fact that, chaotic systems are sensitive to initial conditions [11], and through a unique correspondence to the dynamics, such systems can be utilised to realise a dense set of points that are input to it [12], [13]. Even though the evolutionary dynamics of chaotic functions in itself is very complex, the hardware assembly of chaotic systems is believed to be fairly straightforward [14], [15], as such systems are governed by simple mathematical equations and therefore implementing these systems in hardware domain becomes resource-saving. However successful implementation of a complete stand-alone chaotic measurement system is still under investigation, as hardware implementation of a mathematical function is subject to several non-idealities that affect the circuit performance, such as noise, parameter anomalies due to drift and offsets of the components used.

The primary issue that significantly affects the outcome of a chaos based measurement system is with the inability to maintain ideal parametric conditions in the implemented map function. Since the dynamical behaviour of a chaotic system is strictly governed by the control parameter of the function [16], a slight variation in the parameter can have a huge impact on the evolutionary dynamics and therefore can be responsible for rendering non-ideal chaotic behaviour that may affect the signal measurement utilising it, as was observed and stated by Kapitaniak et al in [17] , Litovski et al in [18], and Sanjin Berberkic in [19]. This work is therefore focussed on studying the dynamical behaviour of chaotic maps with respect to the parametric dependencies, and investigating the methods of estimating the non-ideal parametric value of the chaotic function such that issues

related to the map parameter can be addressed, and the idea of chaotic measurement systems can be brought to reality. In the following section a brief account of chaotic dynamics along with an overview of the measurement system have been presented.

1.1 Chaotic Measurement System: An Overview

Chaotic dynamics is a widely studied area in the field of non-linear dynamical and complex systems. The dynamical nature of a system can be described as time evolution of various states under the influence of parameters that govern the system behaviour [16]. Due to the iterative nature (feedback process) of the evolution, the present and future states of a dynamical system depend on the previous states, therefore a small amount of change in the initial condition or the control parameters may lead to different outcomes and eventually causing the dynamics to digress completely from the expected progression (set of outcomes). Thus, the evolutionary time series (trajectories) of the system may appear to be complex and random-like [11]-[13]. These systems are referred to as chaotic systems which are special cases of dynamical systems that exhibit a pseudo-random behaviour. Despite the apparent randomness, chaotic systems are mathematically well defined and are therefore deterministic in nature. The deterministic principles, hence, aid in estimating the past and future states from the available information, collected over a period of time [13].

Chaotic systems have found use into various applications such as image [20] and data [21] encryption in communication technologies where the chaotic dynamics is used as an identifying signature corresponding to the information that is intended to be encrypted, and can be deciphered only through the complete

knowledge of the chaotic function; that is, the initial condition and parameter used as the cypher key. Also, as has been discussed earlier, chaotic dynamics have found application in signal measurement and considered to be a pioneering approach for analogue to digital conversion [10], [17]-[19]. The fundamental block diagram of the envisioned chaotic ADC can be seen from Fig. 1.2.

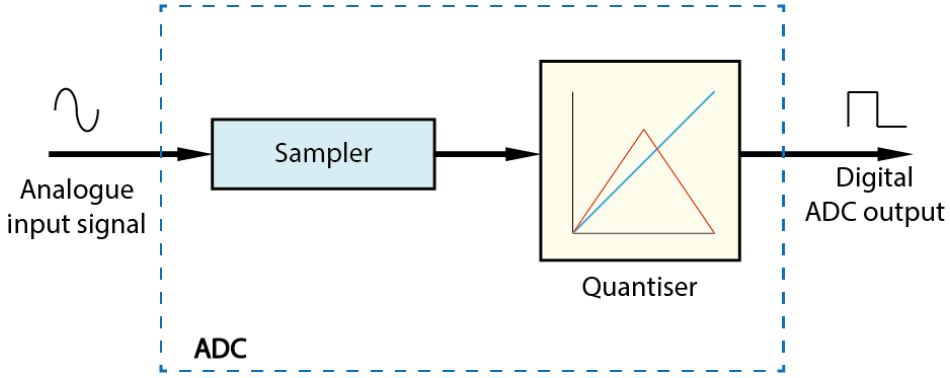


Fig. 1.2 Fundamental block diagram of chaotic ADC

From the perspective of signal measurement, an unknown signal can be input to a physically implemented chaotic function as the initial condition, and the evolutionary dynamics can be generated iteratively by feeding back the outcome of the previous time step as the input for the next iteration. Owing to the sensitive dependence on initial conditions, the resultant dynamics produced by the chaotic map (as an evolutionary time series) may be regarded as the evolutionary footprint holding the key information of the corresponding originating point or the initial condition. Also, when the dynamics is symbolically coded by assigning binary symbols to each of the states of the time series, with respect to a defined threshold, a unique correspondence is observed with the initial condition, as was demonstrated by Metropolis et al. in [22].

For the digitisation of the measured signal, the symbolic sequences produced by the chaotic functions can be of great advantage as the input signals can be

uniquely identified [23]. The iterative dynamics of a chaotic map may therefore be utilised as a symbolic converter (or a quantiser). Determining an input signal fed as an initial condition to a chaotic map is, theoretically, a straightforward numerical exercise as long as the information regarding the dynamical time series is available and mapping function of the chaotic system is known. However, when chaotic maps are implemented in electronic hardware, the map parameters are altered by the imprecision caused due to the offsets and drifts in the hardware components and inherent noise in the electronic circuit [17], [18]. Such parametric alterations may cause the dynamical time series to digress from the ideal one; therefore, mapping back to the initial condition using the non-ideal time series is difficult unless the operating map definition is fully determined in terms of control parameter. Since a small change in the parameter value also introduces great divergence in the dynamics, estimation of the system parameters is necessary on the context of a measurement system.

The chaotic map that is preferred for the application of signal measurement is the Tent Map (TM) [24]. Therefore, the parameter estimation of the TM is performed in order to retrieve sufficient amount of information regarding dynamical behaviour of the system, and hence is the main focus of this work. The mapping property of TM within the state space show uniform distribution over a wide range of parameter values, and dense collection of unique points within the state space can be identified through the corresponding chaotic dynamics. Such uniform chaotic distribution is established by the properties of robust chaos described by Banerjee et al in [25].

Arroyo and Alvarez in [26] have proposed that the symbolic sequence produced by the TM are Gray codes and described a straightforward technique that involved converting the Gray codes to binary numbers and then to decimal numbers to estimate the real valued initial condition from the generated symbolic sequence. However, such techniques did not consider the effects of parametric imperfections introduced by the physically implemented chaotic function.

In case of TM, when the parameter value is reduced the height of the map is reduced causing the dynamics to digress from the desired or actual time series. When Gray code sequences generated by the reduced height TM are directly converted into the corresponding decimal values, it leads to an incorrect mapping and therefore measurement accuracy is greatly affected, as observed in [18]. In [27], some analysis has been presented on the theoretical context mentioning that it is preferable to maintain the map parameter at ideal values. The inevitability of the parametric reduction cannot be ignored when the map is implemented in physical domain, as it is relevant for the case of a measurement system. Kapitaniak et al. in [17] have also attempted to measure electrical signals in a similar way and observed that the dynamics of the physically implemented TM is greatly affected by the component tolerances and offsets. They have shown that, when sequences generated by the physically implemented chaotic map are converted to real values directly, without considering the non-idealities, the estimated outcomes contained significant errors that prevented the outcomes to map correctly to the actual initial condition.

Alternatively, Cong et al. in [28], have theoretically proposed that if the non-ideal value of the map parameter is known, proper estimation of initial conditions

through backward tracking of the iterates over the time series can be performed by operating inverse maps on the timeseries. A recent research [24] has also discovered that a reduced height TM (due to non-ideal parameter) exhibits asymmetric partitioning of the state space over the iterations and the intervals are created in unequal sizes. It has been proposed that, in order to determine the initial condition correctly, the non-ideal value of the parameter must be used to determine the amount of shift in the partitions and correct interval for the initial condition can be determined by accordingly reinstating the partitions: further details regarding this phenomenon has been provided in Chapter 2. It is therefore realised from the available literature that, when chaotic functions are implemented physically, the deviation of the map parameter from the ideal values is inevitable and must be determined in order to estimate the initial condition with reasonable accuracy. In this work, methods have been devised to estimate the control parameter value of the non-ideal TM that results from the physical circuit implementation of the function. The aim of the research is detailed in the following section, and it has been broken down into the following objectives, which will be addressed in the upcoming chapters.

1.2 Aims & Objectives

The aim of the work is to determine the control parameter of the TM operated in non-ideal or reduced parametric conditions. To determine the map parameter, the following objectives must be achieved.

1.2.1 Realising the Dynamics of TM Over Various Parameters

The dynamical properties of the TM must be understood clearly over a range of parameters. From the state space distribution of the chaotic dynamics, the

relationship between the map parameter and the dynamical limits namely maximum and minimum should be observed and analysed, such that the limits can be utilised as an indicator of the map parameter.

1.2.2 Realising Symbolic Correspondence with the TM Parameter

It is important to symbolically define the map trajectories and establish their correspondence with the iterates and the map parameter. How partitions and subintervals are created and shifting of partitions from the ideal positions in case of reduced parameter should be observed in order to establish a relationship between the symbolic sequences corresponding to the map maximum and the minimum.

1.2.3 Studying the Behaviour of the TM Fixed Point

The non-zero fixed point of the map is where the $x = y$ line intersects the map whose value changes with the change in the parameter. As a result, identifying the fixed point and quantifying its value can be related to the value of the map parameter.

1.2.4 Investigating the Noisy Dynamics of TM

The dynamics of the TM needs to be further investigated considering noisy conditions. The noisy trajectories interact uniquely with the map fixed point and the information regarding the non-zero fixed point remains to be preserved through the dynamics and therefore can be utilised to determine the map parameter.

1.2.5 Determining the Methods to Estimate the Parameter

From the available knowledge of the TM dynamics (both symbolic and noisy dynamics), suitable methods to estimate the map parameter must be formulated. The proposed methods must be validated through numerical simulations and detailed analysis of the results.

1.3 Original Contributions

In this work, a broad study has been conducted to estimate the dynamical parameter of the tent map (TM). The observations, analysis and the estimation methods along with the results are the original contributions made for the development of this work and culminating into the thesis. Following are the key contributions made in the field of dynamical systems and chaotic measurement:

- The parameter estimation method from the symbolic code of the dynamical maximum (description published in Section IV of the article [24]) using symbolic shifting window has been contributed.
- Further, a difference equation has been formulated from the code of the map maximum that directly solves for the parameter. The equation is based on the newly discovered relationship (or differences) between the ideal and non-ideal symbolic codes in terms of the map parameter. This contribution has been explicitly detailed in Section 4.1 of this thesis.
- Through the development of this work, the crossovers in the samples of noisy time series have been first observed and presented in Section 3.1.3. Also, the observations have been published in Section 3 of the article [29].
- It has also been established that the observed crossovers correspond with the map fixed point and a method has been proposed to determine the

parameter from those crossovers (for details see Section 4.2). The method has also been presented in Section 4 of the article [29].

- Original results have been produced using numerical simulations (through MATLAB R2016b) and detailed analysis have been presented in Chapter 5. Where the estimated outcomes (parameters) proved to be promising, using only 200 iterations (as the length of the trajectory), with errors below 1%.
- In the noise oriented approach, the map parameter has been recovered with such an accuracy for signal to noise ratio 15-30 dB and onwards.

1.4 Organisation

The work is organised as follows, in Chapter 2, Sections 2.1 – 2.9 the background information regarding dynamical properties of TM have been described followed by establishment of the symbolic dynamics and its correspondence to initial conditions and subintervals. The chaotic distribution of the system states has been studied in great detail and the definitions of map maximum and minimum have been established along with the corresponding symbolic identities.

In Chapter 3, several challenges have been discussed that are encountered while the map is implemented in physical hardware domain. The effect of parametric reduction has been observed and studied from the perspective of initial condition estimation. The impact of noise in the chaotic dynamics is explored and how the properties of fixed point can still be useful for the extraction of meaningful information have been discussed. The available knowledge regarding realising the symbolic sequence in terms of the initial condition and parameter estimation

has been elaborately analysed and the inadequacies of the conventional approaches have been identified.

In Chapter 4, some solutions to the parameter estimation problem have been presented in the form of computing algorithms. The algorithms independently consider both symbolic and real valued platforms for more realistic cases e.g. noisy dynamics. The proposed algorithms have been described in detailed steps such that in can be easily implemented in the processing domain.

In Chapter 5, the proposed algorithms and the effectiveness were analysed using mathematical simulation and graphical results.

In Chapter 6, the work has been concluded in terms of the knowledge gained and solutions offered to solve the problems. Also, delivering the proper understanding considering the implementation of the techniques along with some further proposals as future work to meet the remaining technological needs such that the chaotic measurement system can be made possible in reality.

2 BACKGROUND LITERATURE REVIEW

In this chapter, the theory of measurement system is presented followed by several ADC architectures whose design complexities and shortcomings have been briefly discussed. Then chaotic dynamics and behaviour of chaotic maps have been broadly discussed from the perspective of the system parameter, delivering the insight for the realisation of suitable parameter estimation techniques in the upcoming chapters.

2.1 Measurement Theory

A typical electronic measurement system involves a sensory device, an amplification and signal conditioning stage, an ADC and a microprocessor. A sensor is a material or a device that can respond to changes in the physical states in the form of electrical signals. A wide variety of sensors are available and can be chosen according to the type of the physical quantity to be measured. To enhance the amplitude and quality of the sensed signal, further amplification and conditioning stage is introduced. The signal amplification is performed by electronic amplifiers and conditioning of any noise is performed by electronic filters. Finally, an ADC is used to measure and quantise the analogue electrical signal into digitised signal. Through further incorporation of microprocessors, the digitised signal is represented numerically, stored, displayed or can be processed for some decision making.

Measurement systems always have some errors and tolerances that are responsible for the uncertainty in the measured quantity. Measurement uncertainties are generally categorised into systematic and random errors. The

systematic errors are consistent deviations in the measurement, that occur due to the definite causative factors such as the inaccuracies in the calibration and the transfer function in the signal conditioning stage of the measurement systems. The random errors are fluctuations in measurement around the actual value which is mainly caused by induced noise in the system. There are several factors that govern the quality of measurement which are: accuracy, precision and resolution.

2.1.1 Accuracy, Precision and Resolution

The term accuracy [1], [30] identifies how close the measured outcome is from the actual value. The uncertainty in the measurement given by the difference between the measured and actual values is usually dependent upon the two sources of errors: one is the measurement error or the uncertainty in the reading, and the other is error relative to the full scale of measurement [31]. The measurement error is generally caused due to the tolerances of the components used in the measurement system. This type of error can also occur during digitisation of a signal, since tolerances in the voltage dividers might affect the reference values, leading to deviation in the reading. The error values are specified in percentage or parts per million (ppm). The total absolute uncertainty ($Total\ Error$) [1], [32] due to these errors is determined by the additive sum of the error in the measured reading ($Measurement\ Error$) and the $Full\ Scale$ offset error:

$$Total\ Error = \frac{1}{100}(\%\ Measurement\ Error \times Reading + \%\ Full\ Scale\ Error \times Range)\,, \tag{2.1}$$

where $Reading$ is the absolute measured outcome, and $Range$ is the scale within which the measurement is to be taken. The $Full\ Scale\ Error$ can therefore be

defined as an error expressed as the percentage of full scale of measurement range, which signifies that a reading will belong within the error bounds $\pm\, Full\ Scale\ Error\ (\%) \times Range$. The error percentage in terms of accuracy is given by:

$$\%\ Accuracy\ Error = \frac{Total\ Error}{Reading} \times 100\ . \qquad (2.2)$$

The accuracy error can hence be treated as a systematic error caused due to the tolerances or offsets and anomalies in the transfer function or gain in the measurement system. Apart from offset errors, there can be other forms of inaccuracies, for example, scaling error and nonlinearity.

Often, in technical fields, the terms accuracy and precision are used interchangeably. However, each of these terms can be defined independently. The random error or deviations caused by the noise in the measurement system is indicated by the term precision [30].

The noise affecting precision are mostly thermal noise and electromagnetic interferences in an electronic measurement system. The random noisy spectrum exhibits a Gaussian or normal distribution when distributed over a range [1], [30]. If a histogram is obtained from sufficient collection of random variables, a heaped bell curve about the mean of all the random points is observed corresponding to the peak of the curve.

The random error in the system can be removed by filtering (High pass, Low pass, Band pass) [1] or by oversampling of a reading and averaging of the samples, as in the practical scenarios, it is highly likely that a large set of randomly distributed samples have its mean close to the actual signal that is

intended to be measured. Hence filtration and averaging techniques can be utilised to improve precision.

Another aspect of measurement system is resolution [1], [31] which is the smallest measure of change that can be numerically realised by the system. The change detected by the measuring device is usually expressed as a point defined by the number of bits within a range. Therefore, a measurable range can be represented through steps of change that is of the smallest possible magnitude the system can measure. Generally, the term resolution is associated while converting the analogue signals into digital equivalent using the ADCs. Following are the calculations to determine the resolution of a measurement system shown through an example. Assuming a measurement system that can measure a ±5V range (10V span) using a 16-bits A/D converter. There are up to 2^{16} = 65536 points that can be defined by a 16-bit digital code. Therefore, 1 part of the 65536 points within the span of 10V, i.e. 10V ÷ 65536 = 152.5 microvolt (uV) can be detected as the smallest size of the change or increment by a 16-bit ADC.

2.2 Analogue to Digital Conversion

Signal measurement and conditioning is predominantly performed in digital computational systems, therefore, ADCs form the most integral block between the analogue and digital domains. The basic principle of ADC operation is based on comparison of the input signal with a reference, dividing the range into levels of equidistant step size and generating an equivalent numerical value [2], [31]. The number of steps is determined by the step size or the resolution of the ADC. Since the resolution depends on the number of bits, each step size is identified by

a combination of bits. Therefore, for an n bit ADC, the step size (Z) with maximum input of V_{ref} is given by equation (2.3).

$$Z = {V_{ref}}/{2^n}. \tag{2.3}$$

As can be seen from Fig. 2.1, the small change in signal is detected by the finer step size but not the coarse one. However, increasing the number of bits also increases the components and therefore the complexity in the circuitry. As a result, the conversion speed of the ADC is greatly affected. Conversion speed of an ADC is determined by the time taken by an ADC to identify the signal level and generate equivalent binary outcomes. The conversion speed is dependent on the sampling frequency i.e. the number of samples collected within a second.

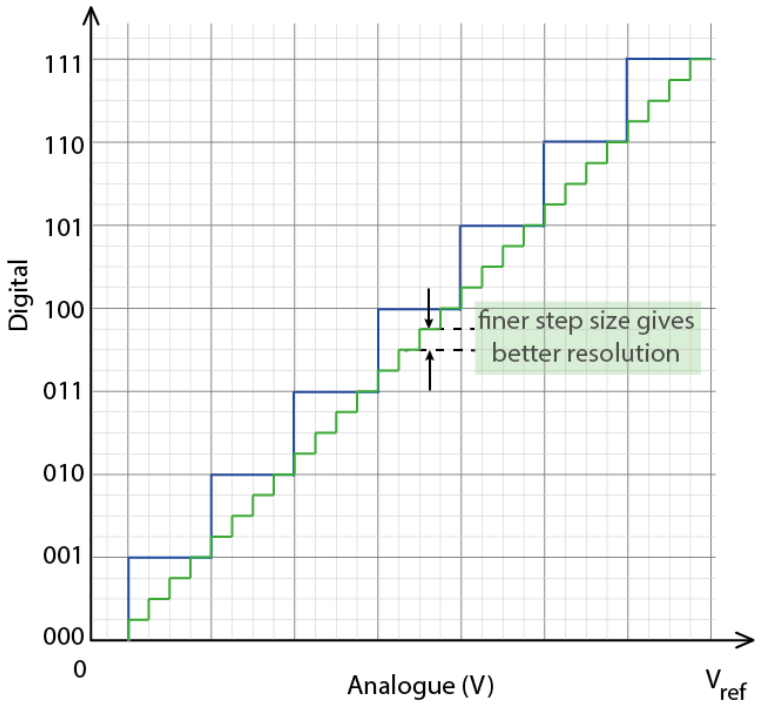


Fig. 2.1 Reducing the step size improves resolution

In order to avoid the loss of signal changes, the sampling frequency must always be maintained at least twice or higher than that of the bandwidth of the input signal. This principle is commonly referred to as Nyquist criterion [2]. Any

sampling at the rate less than the Nyquist rate results in void between the samples due to the inability to register the changes between the two samples. This phenomenon is termed as aliasing. Therefore, in order to avoid aliasing, for an ADC with a sampling frequency f_s, the signal bandwidth f_B must be maintained within half of that of f_s i.e.

$$f_B \leq 0.5 f_s. \tag{2.4}$$

Depending on these criteria and the resources that are used to implement an ADC, with varying conversion times result in development of a number of different ADC architectures, each of them prioritising a different criterion.

Following is a brief discussion of the major types of ADCs – among which the Flash, Delta-Sigma (ΔΣ), Successive approximation register (SAR), pipeline and the hybrid flash types are most common. The simplest and the most basic ADC is the Flash type converters [3] in which, the resolution is determined by the number of segments that the input signal is divided into by the voltage dividers. Therefore, every time the resolution is increased by one bit (for n+1 bits) the number of comparators get doubled (2^{n+1} comparators). Beyond 6-bits, the number of comparators required results in significantly increased chip area, whereas commercial devices usually require at least 8-bit conversions.

Compared to flash ADCs, ΔΣ type ADCs [4] consists of a single bit digital to analogue converter (DAC) which acts as the Δ sub-circuit and produces a threshold voltage level equivalent to the single bit resolution of the input. Once the threshold is achieved, a pulse is generated; therefore, the frequency at which the pulse is generated depends on how often the threshold value is reached.

The SAR [5] type ADC consists of a control register that is used to generate the reference data on each conversion using a DAC and compared with the actual input. Depending on comparison, the previous bit in the register is updated and thereby the equivalent digitised data for the input signal is produced.

In a pipelined ADC [6], the conversion process is broken into several smaller conversions. Each stage converts the outcome of the previous stage (input signal for the first stage) into coarse grained digital equivalent. The outcome is then scaled up and converted again through a DAC for the next stage of comparison and the process is continued to generate the bits.

To reduce the number of preamplification units in flash ADCs, interpolating stages [7] (more voltage dividers) are added, therefore the architecture is regarded as hybrid flash. In order to improve the quality of the measurement, a folding stage [33] is often included using which the input is folded into smaller range. A combination of a coarse and a fine ADCs determines the range within which the folded input belong and accordingly the input is digitised through this comparison.

As can be realised from these ADC architectures, the quantisation block is modified and combined to improve various aspects of the ADC parameters, depending on the priority of the application. This is usually performed through additional circuitry (like folding or interpolating) or by increasing the number of comparators or other components like DAC, and coarse and fine ADCs. The resolution the ADCs can only be improved at the cost of more components. As a result, either power consumption or chip area or circuit complexity is maximised. In order to optimise these factors, other possibilities and techniques must be

considered to replace the quantisation technologies that are commonly used in the available architectures.

2.3 Chaotic Dynamics and Definitions

Chaotic maps can be chosen as a quantisation block in a measurement system because such maps are simple mathematical functions which are easily implemented with fewer resources. Also, in order to increase the resolution, the same function can be operated iteratively through a feedback, without having to redesign any additional hardware. However, given that chaotic maps are highly sensitive to both initial states as well as map parameters, system errors introduced by the physical implementation play a major role in the behaviour of the map dynamics [17], [18]. Therefore, to implement a chaos-based measurement system, an algorithmic approach must be adopted. Such algorithms are heavily reliant on the map parameter, which must be recovered accurately in order to successfully implement the measurement system. Given the nature of the problem, the recovery of the map parameter becomes a pre-requisite in developing the algorithm for signal measurement using chaotic quantisation system.

Dynamical systems can be described by the evolution of the various states over time under the influence of the factors that govern the system behaviour [34], [35] and can be mathematically defined by equations. Involvement of several factors governing the dynamics can lead to a complex behaviour whose evolutionary states on observation may appear to be random. Chaotic systems [11]-[13], are special cases of dynamical systems that exhibit a pseudo-random behaviour, despite that, these systems are mathematically well defined and are

therefore deterministic in nature. Since the current state of a system is responsible for the next states, evidently, all the future states retrospectively depend on the previous states, and hence, a small amount of change in the initial condition can lead to highly digressing or different future outcomes. The evolutionary dynamics of chaotic systems are so sensitive to the initial condition and the control parameter of the system that long-term predictability of the future states depend on the precision and accuracy with which the current states are determined [34].

Chaotic dynamics can be defined as a function $f(x,\mu)$ of control parameter μ and the input x which can also be referred to as the current state. The dynamic evolution of a chaotic system can be studied through time series representation of the system states given by: $\mathcal{X} = \{x_n \mid n = 0, 1, \dots, N-1\}$ where $\mathcal{X}$ is referred to as a trajectory or the orbit of the dynamic process containing N number of states starting from an initial condition x_0 [13]. As the future states depend on the current state, the iterative process is of feedback type, where a single iteration of the function is performed by inputting the outcome of the current stage x_n to determine the outcome of the next stage as given by $x_{n+1} = f(x_n,\mu)$. Hence, the trajectory of an initial condition with N dynamical states (or iterates) can be represented as x_0, x_1, x_2, …, x_{N-1}. The iterates in the trajectories can also be mapped within a range of possible outcomes that the system can generate; such a range is often called phase-space or state-space [13].

There are two essential criteria that are necessary for the evolutionary dynamics to be chaotic; these are the stretching and folding operations exhibited by the chaotic functions. The stretching behaviour of the dynamical system is

responsible for the evolutionary trajectories to be divergent even though two neighbouring points are separated by very small distance between each other. Such divergence was first analysed by Aleksandr Lyapunov, who proposed that when two originating points are separated by a negligible distance, over an iterative dynamical evolution, the trajectories of the two points will gradually become divergent from each other [36], [37] since the distance of separation between the two points will exponentially increase over iterations. The exponential divergence is therefore analysed as a rate by which the two points deviate from each other on every iteration and is termed as Lyapunov exponent (λ) given by

$$\lambda = \ln(|p_{n+1}|/|p_n|), \tag{2.5}$$

where p_n is the small distance in a close neighbourhood of the actual point x_n resulting into a deviation $x_n + p_n$.

Due to the stretching nature, the dynamical iterate given by $x_{n+1} + p_{n+1}$ is diverging away from the actual x_{n+1} when $p_{n+1} > p_n$. The folding nature of the chaotic systems plays a salient role in confining the dynamics within the state space, as just the stretching nature alone would have caused the dynamics to escape to infinitely incrementing trajectories. Both to the stretching and folding nature is therefore, responsible for dense mixing of trajectories within the state space and therefore making the dynamics sensitive to initial condition.

The mapping of the trajectories is studied both numerically and graphically for further analysis. In the following sections more of such graphical views and

corresponding mathematical description of various chaotic systems have been presented.

2.4 Chaotic Maps

Relating to the structural and dimensional features, there are several ways to classify dynamical systems. The chaotic maps are usually classified as unidimensional or multidimensional systems depending on the univariate or multivariate mapping of the system states as defined by the map function. A few of the various dimensional chaotic maps that are widely popular in the field of dynamical systems, for instance, a three-dimensional (3D) chaotic map: Lorenz system [34], two-dimensional (2D) chaotic maps: Hénon map [12], and one-dimensional (1D) chaotic maps [12], [13]: Logistic Map (LM), Bitshift Map (BM), and Tent Map (TM) that have been discussed briefly in the following parts of this section. The 2D and 3D maps are difficult to achieve electronically because the parametric relationships to the transfer functions are too complex to achieve. Therefore, 1D maps are widely embraced for the simplicity and implemented on the context of applied chaos in physical hardware.

In the following sections the properties of various 1D chaotic maps have been broadly discussed and analysed through simulations of bifurcation diagrams, time series plots and function plots. For these observations, the math processor MATLAB R2016b has been used (alternatively the open source software Octave can be used), where the programs for each observation and graphs have been provided in the respective appendix as referred to in the sections.

2.4.1 Logistic Map

Out of all other chaotic maps, 1D maps in particular have gained special attention [12], [13] because these systems exhibit the most fundamental type of chaotic behaviour and can offer a wide level of complexity under various parametric conditions. The majority of the 1D maps, apart from BM, are *unimodal* as these maps contain a well-defined unique *peak* or *maximum* in the *topological structure* e.g. LM, TM. The LM is mathematically expressed as

$$x_{n+1} = 4rx_n(1 - x_n), \tag{2.6}$$

where, r is the parameter value ranging from [0,1] that controls the height of the map, and x_n and x_{n+1} are respectively the current and future states of the system. The LM function (refer to Appendix 2.1 for program) is shown in Fig. 2.2.

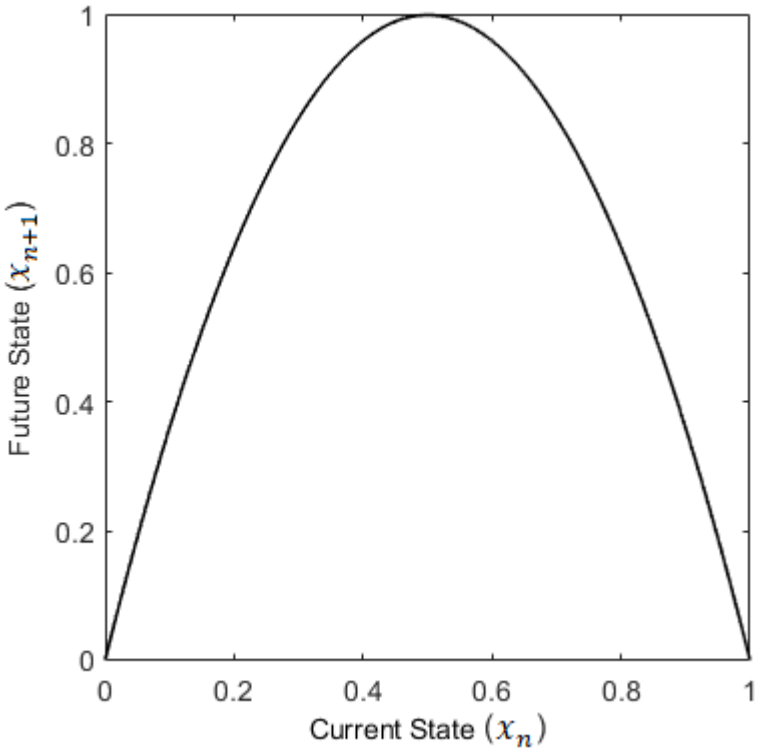


Fig. 2.2 Logistic Map

The LM gained its popularity during a demographic study, when the dynamical model of population growth was analysed by Pierre Francois Verhulst [38]. The LM dynamical model is widely applied to understand population evolution, species conservation, cycles in predator prey model. One of the useful ways to study chaotic maps is to graphically plot the distribution of the function states against the control parameters, commonly referred to as bifurcation diagram. When parametric dependencies of a system need to be investigated, it is essential to study the bifurcation diagram of the map [13], [16], because through these diagrams, the mapping behaviour of the system states within the state space can be analysed in terms of periodicities, bifurcations, chaotic distributions, and the upper and lower limits (maximum and minimum) of the distribution against different parametric conditions. The bifurcation diagram of LM has been shown in Fig. 2.3 (generating program available in Appendix 2.2).

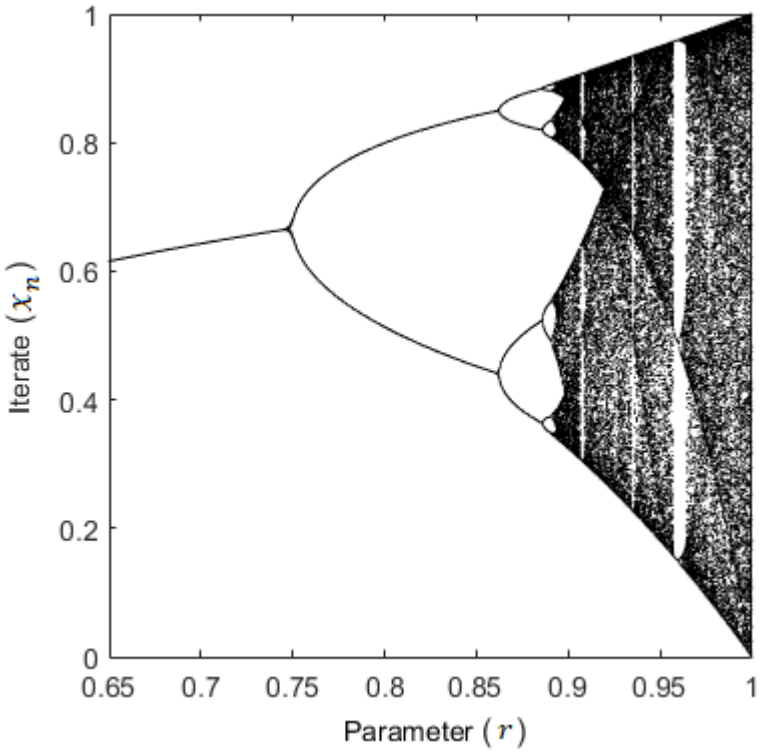


Fig. 2.3 Bifurcation diagram of Logistic Map

The bifurcations and windows of periodicity for a range of parameter values can be observed. For parameter value up to r = 0.75 the map shows a fixed-point behaviour, periodicities can be noticed in the dynamics with intermediate chaotic bands up to $r \approx 0.9571$ where the typical period doubling nature of the dynamics can be clearly observed between parameters approximately $r \approx$ [0.75,0.89], forming windows of periodicity in the distribution.

The LM has also gained a lot of attention in engineering applications – e.g. chaos control and synchronisation – and also pseudorandom number generation in encryption and keying in the area of communication [39]. For the desired application of measurement, the LM might not be a proper choice of the chaotic map as for several parametric conditions the map is periodic and generation of unique chaotic trajectories for a set of initial conditions will be difficult.

2.4.2 Bitshift Map

Also known as the Bernoulli Map, BM is a piecewise linear (PWL) 1D map which is of type bimodal function as BM has two peaks defined over the state space. The following is the mathematical definition of BM [13],

$$x_{n+1} = \begin{cases} 2\mu x_n & 0 \le x_n \le x_c \\ 2\mu x_n - 1 & x_c < x_n \le 1 \end{cases}, \tag{2.7}$$

where, μ is the parameter ranging from [0,1] depicting the map height and x_n and x_{n+1} respectively are the current and future states. As can be seen from Fig. 2.4 (generated by program in Appendix 2.3), the BM has two stretching sides separated by a midpoint x_c = 0.5.

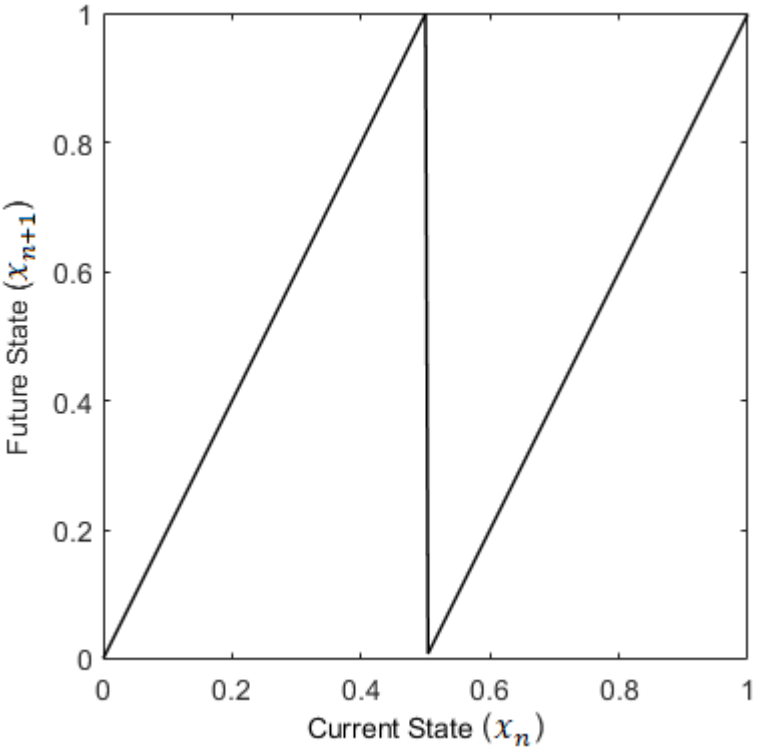


Fig. 2.4 Bitshift Map

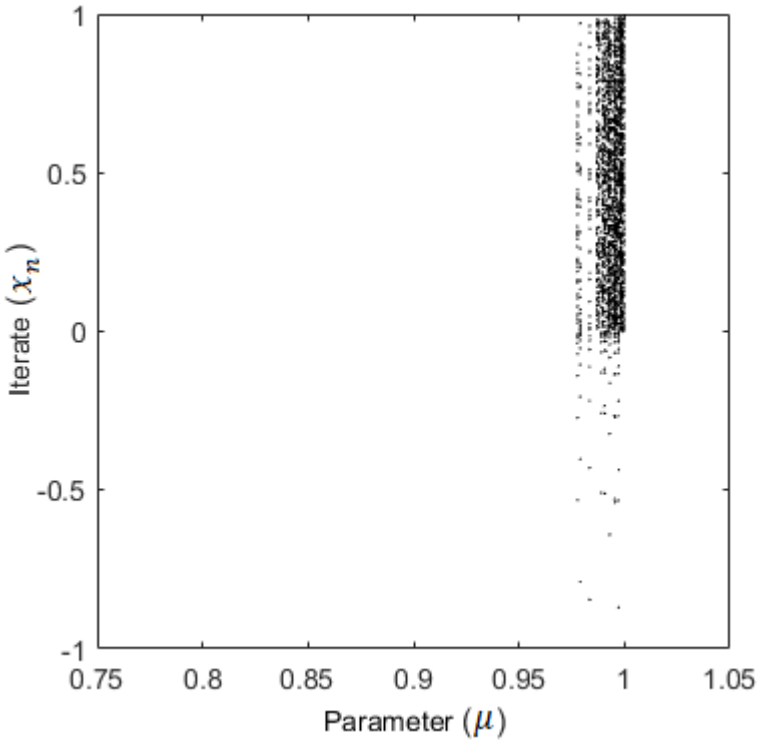


Fig. 2.5 Bifurcation diagram of Bitshift Map

Fig. 2.5 is the bifurcation diagram of the BM (see Appendix 2.4 for program); it is understood that the map dynamics, and hence chaos, produced by BM is only

defined for the full parameter value of $\mu = 1$: for any parameter value $\mu < 1$, the dynamics will escape to negative infinity. BM therefore, might not be a suitable map for the desired application where parameters are most likely to deviate from the ideal values due to material challenges and errors in physical implementations.

2.4.3 Tent Map

The other PWL 1D map is the Tent Map (TM) [16] which has received a lot of attention for the simplicity and ease of implementation in electronic hardware domain. The TM (generated by the program in Appendix 2.5 as shown in Fig. 2.6) is defined by a univariate dynamical quantity x over a unit *invariant interval* or the *state space* $I = [0,1] \subset \mathbb{R}$, such that $x \in I$ and a *control parameter* given by $\mu \in [0,1]$.

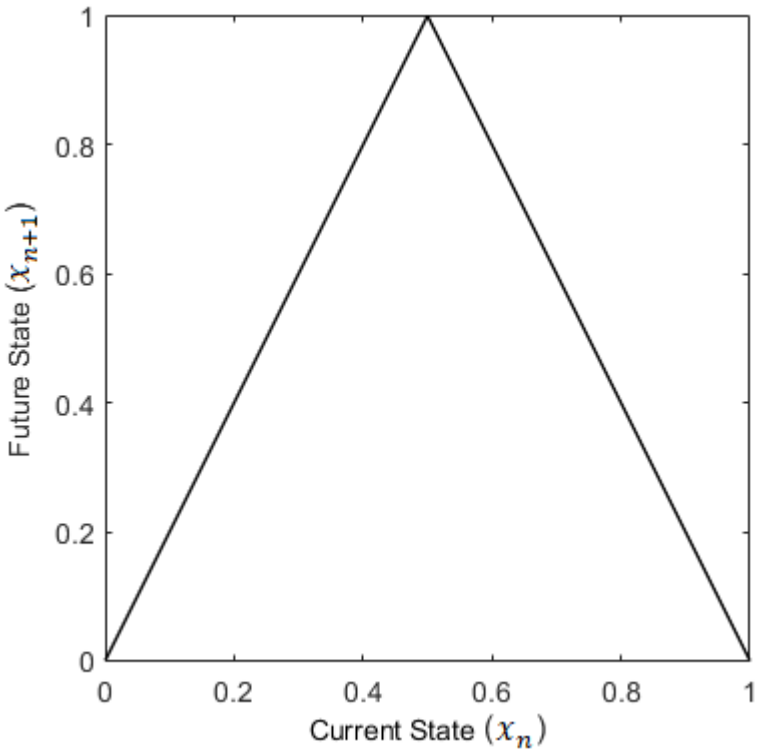


Fig. 2.6 Tent Map

The map is mathematically realised as a parametric self-mapping function $T : I \rightarrow I$ with piecewise monotonically stretching and folding sides about a *critical point* $x_c = 0.5 \in I$. The iterative dynamics $T(x_n) = x_{n+1}$ is defined as

$$x_{n+1} = T(x_n) = \begin{cases} 2\mu x_n & 0 \leq x_n \leq x_c \\ 2\mu(1 - x_n) & x_c < x_n \leq 1, \end{cases} \tag{2.8}$$

where x_n and x_{n+1} respectively are the current and future states and n is the time step index for the dynamical states of the TM. The stretching and folding nature of the TM causes the points in the invariant interval $I \subset \mathbb{R}$ to *eventually* map arbitrarily close to each other [13] and hence, dense distribution of points is obtained over I for a wide range of parameter values. In Fig. 2.7, the bifurcation diagram of the TM (generated by the program listed in Appendix 2.6) has been shown for the parameter $\mu \in (0.5,1]$.

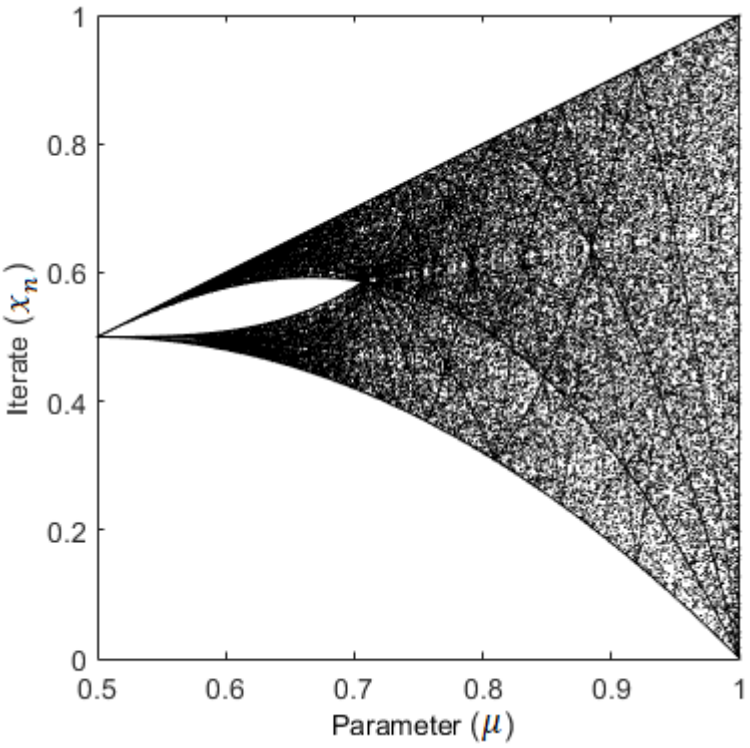


Fig. 2.7 Bifurcation diagram of Tent Map

Unlike the LM or BM the dynamics of TM for any initial condition $x_0 \in I$ is chaotic over a wide range of parameter values, and as can be verified from the bifurcation diagram, the distribution for $\mu \in [0.707,1]$, has no prominent windows of periodicity. Such a dense distribution of the chaotic trajectories is also known as 'robust chaos' [25] in which case the dynamical states are unique, attributed by a pseudorandom behaviour with no mere repetitions or periodicities in the trajectories. This is the reason why the TM is so widely used in the applications like random number generation, encryption, cipher key generation in the area of communication and image processing technologies [20], [21], [40].

For the desired scope of application in the area of measurement system, it is of primary interest that a wide range of points must be evaluated, for which the TM can be chosen as a suitable candidate as uniquely dense distribution of points can be realised through the dynamics of TM.

In the following sections the properties of the TM have been further discussed in detail to provide a clear view in the subsequent sections regarding how the map is utilised for the intended application.

2.5 Properties of Tent Map

The iterative discrete time trajectory of an input or *initial condition* $x_0 \in I$ through $T(x_0)$ can be defined as $\mathcal{X} = \{x_n \mid n = 0, 1, \dots, N-1\}$, with N iterates. Hence, the dynamical properties of the TM can be described as [26]

1. $x_0 = T^0(x_0)$
2. $x_{n+1} = T^{n+1}(x_0) = T(T^n(x_0)) = T(x_n)$
3. $T(0) = T(1) = 0$

4. $T_{max} = T(x_c) = 2\mu x_c = \mu \leq 1$, where T_{max} is the maximum height and the dynamic maximum of the map, for $0.5 < \mu \leq 1$
5. $T\,(T_{max}) = T^2(x_c) \geq 0$, $T(T_{max})$ is the dynamic minimum reached over a long-term iteration.
6. $x_f = T(x_f) = 2\mu(1 - x_f)$, where x_f is the non-zero fixed point [13].

As can be seen from Fig. 2.6 and equation (2.8) of the TM, the map constitutes a positive as well as a negative slope on either side (left and right respectively) of the critical point x_c. The negative slope is responsible for the reversal of the map behaviour resulting in the dynamics being folded whenever the condition $x_c < x_n \leq 1$ is realised by the TM. Any point exhibiting the dynamics as $x_{n+1} = x_n$ is known as a fixed point. There are two fixed points of the TM in the state space I which are shown in Fig. 2.8.

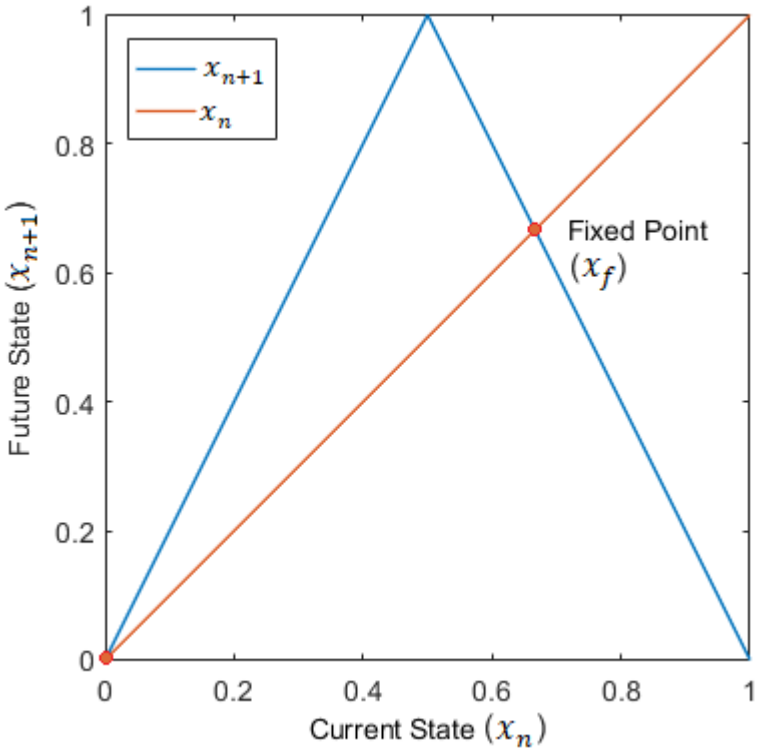


Fig. 2.8 Fixed point of the ideal TM

One such fixed point is $T(0) = 0$. The other is the non-zero fixed point $T(x_f) = x_f$, given by:

$$x_f = 2\mu/(1 + 2\mu). \tag{2.9}$$

The x_f has direct correspondence with the map parameter μ, meaning as μ is varied, the x_f shifts gradually along the $x = y$ line (map diagonal $x_n = x_n$), as can be seen from Fig. 2.9.

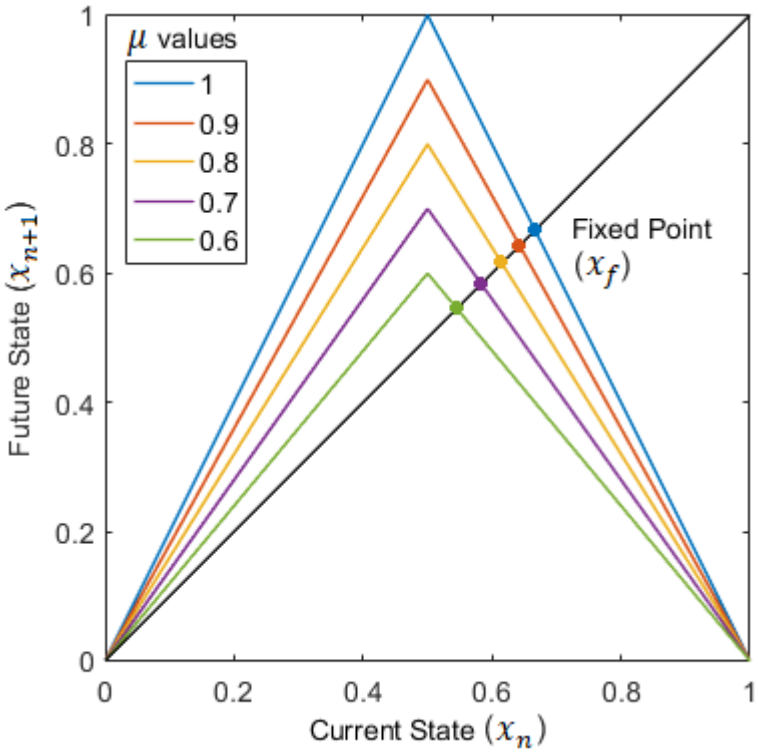


Fig. 2.9 x_f moves along the map diagonal as the parameter is altered

Observing the equation (2.9) across the parameter μ, the change in the x_f can be seen in Fig. 2.10. The chaotic dynamics produced by the TM due to the stretching and folding result into constant shuffling and mixing of the state space I [12], [41]. Such shuffling and mixing mainly occur about the non-zero fixed point [35].

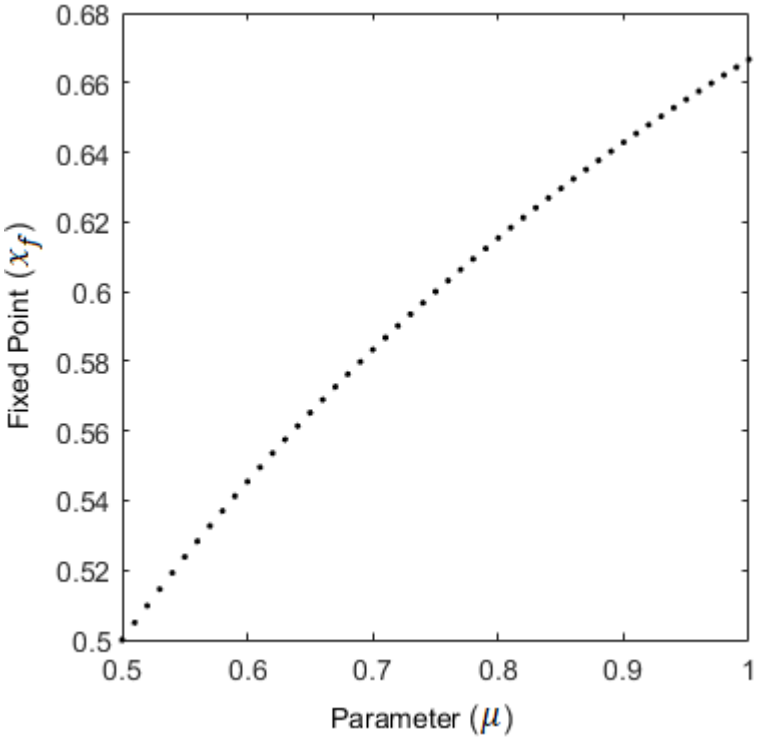


Fig. 2.10 Change of x_f with respect to the parameter

2.5.1 Maximum and Minimum of TM Trajectories

In practical implementations, the parameter may not be maintained constant at the ideal value μ = 1. Under such non-ideal conditions, when μ < 1, certain changes in the dynamical characteristics of the map may be noticed relating to the distribution of points or the attractor. The property of the dynamical attractor of the TM can be defined by the dynamical maximum and minimum that are expressed as a function of the control parameter μ [16]. If a value of x_c = 0.5 is reached at any state in a dynamical trajectory, the next iterate will immediately map onto the maximum $\mu = T(x_c) = 2\mu(0.5)$. Once the maximum is reached ($x_n = \mu$), from the second restriction of the TM equation (2.8), $x_{n+1} = 2\mu(1 - x_n)$ for $x_c < x_n \leq 1$, substituting x_n with μ, the next iterate will therefore map onto $2\mu(1 - \mu)$, which is the minimum value that an iterate can reach [16]. Hence, the maximum (T_{max}) and minimum (T_{min}) are respectively defined as:

$$T_{max} = T(x_c) = \mu. \tag{2.10}$$

$$T_{min} = T(T_{max}) = T(T(x_c)) = 2\mu(1-\mu). \tag{2.11}$$

If sufficiently long-term dynamics are studied, it can be observed that the trajectories of arbitrary points originating from I are *eventually* gravitating to be trapped within a boundary I':

$$I' = [T_{min}, T_{max}] = [2\mu(1-\mu), \mu], \tag{2.12}$$

where $I' < I$, when $\mu < 1$ [16], as it can be verified through cobweb diagrams. The trajectory (of $N = 300$) of an initial condition originating below the T_{min} for $\mu = 0.75$ is shown in Fig. 2.11.

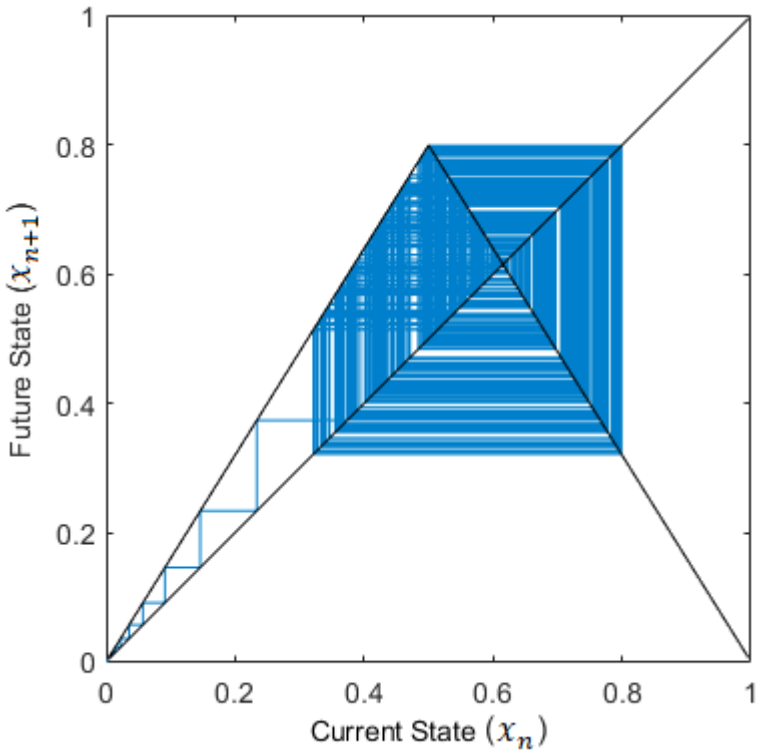


Fig. 2.11 Cobweb diagram for $\mu = 0.8$; $x_0 = 0.000124$

Another initial condition, arising from a point beyond T_{max} for the same parametric condition, is shown in Fig. 2.12. Both the plots for the cobweb diagrams are generated by the program in Appendix 2.7.

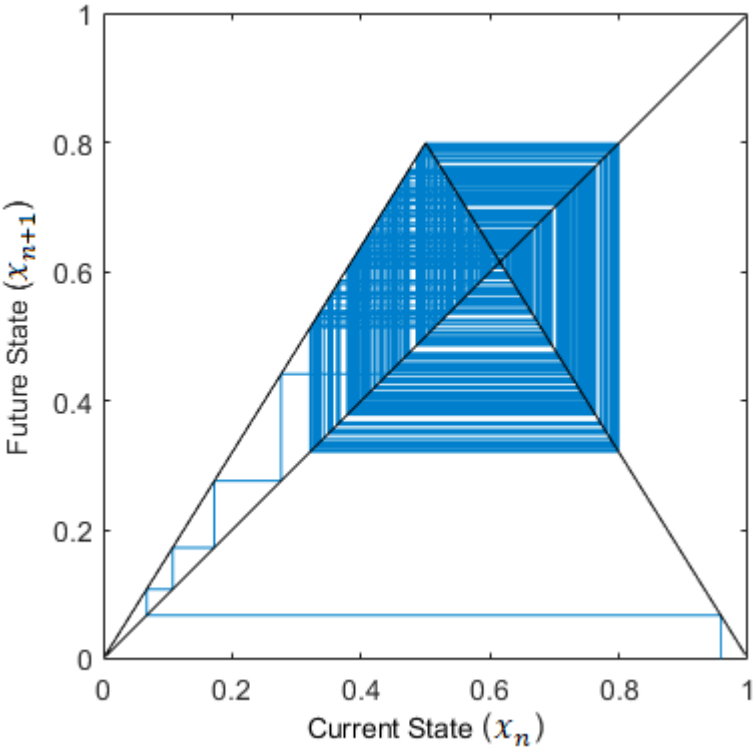


Fig. 2.12 Cobweb diagram for μ = 0.8; x_0 = 0.823

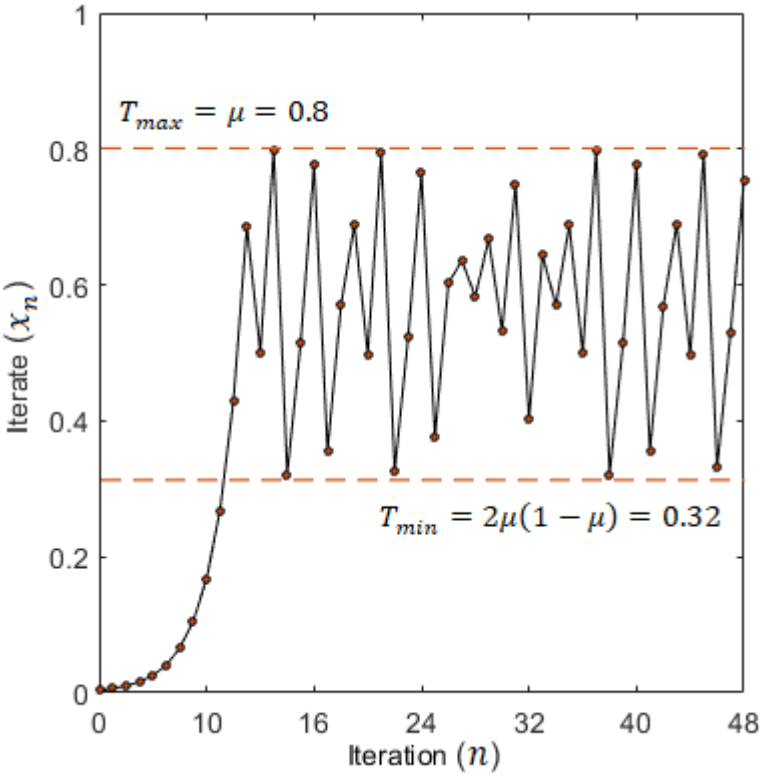


Fig. 2.13 Maximum and minimum of a trajectory with μ = 0.8

The range I', therefore, can be termed as an attractor with its boundaries T_{min} and T_{max}. In Fig. 2.13, The same phenomenon can also be observed through the time

series shown, where the dynamic trajectory of an arbitrary initial condition originating outside I' is attracted to be trapped within the bounds T_{min} and T_{max}.

In Fig. 2.14, the comparison between the maxima and minima for both the cases of $\mu < 1$ and $\mu = 1$ is shown. For $\mu < 1$, the points in the shaded regions are never mapped by any trajectory of the TM once the attractor is visited. As the parameter continues to reduce, the segments of the state space represented by the shaded region gradually increases, implying that the attractor or the range defined by the bounds T_{min} and T_{max} will be narrower.

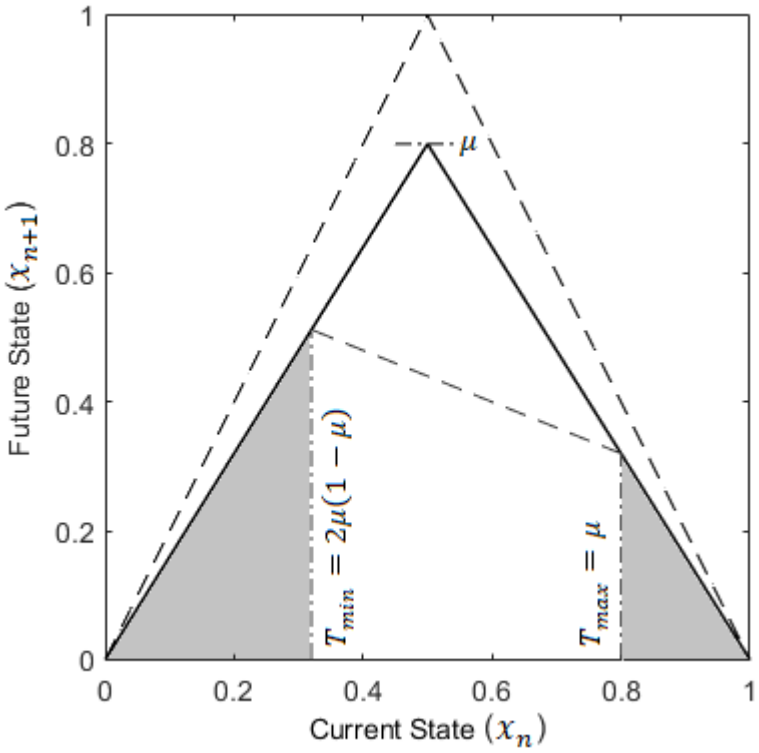


Fig. 2.14 The dynamical attractor I' for a parameter $\mu < 1$

This phenomenon can be further observed across a range of parameter values through bifurcation diagram where, as the μ continues to decrease, the mapping space between the maximum and the minimum also gets reduced gradually. Fig. 2.15 shows the global distribution of the TM dynamics for every μ varying between 0.5–1 with the corresponding maximum and minimum points.

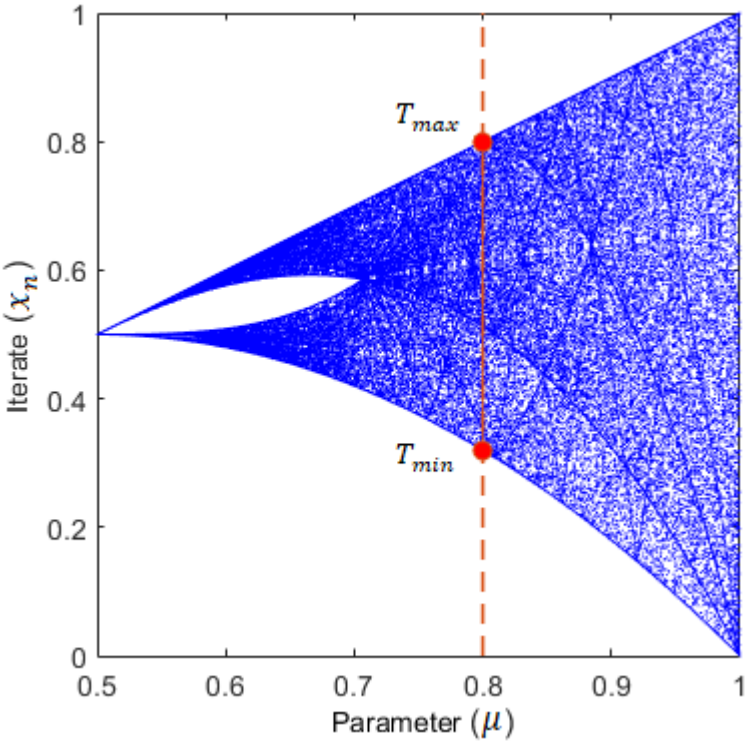


Fig. 2.15 Maximum and minimum over parameter

2.5.2 Symbolic Dynamics of TM

In order to have a better understanding of the dynamics without engaging much resources to record the real valued iterates, symbolic coding of trajectories can be done. The symbolic trajectory is a coarse-grained version of the real dynamical trajectory that was first proposed by Metropolis et al. [22]. At that time alphabetical symbols *L* and *R* were assigned to intuitively indicate *left* and *right* sides about the critical point and symbol *C* was assigned to indicate the *centre* which itself is the critical point of a 1D map. The symbols generated about the critical point (primary partition) also serve as a dynamical footprint that perfectly corresponds to the real valued dynamics of an initial condition.

The alphabetical symbols *L* and *R* was later replaced with 0 and 1 respectively considering the critical point to be one of the edges of the two sides defined by the partition [23], [26]. Hence, the real valued trajectories $\mathcal{X}$ of the TM can also

be transformed into symbolic trajectories $S = \{s_n \mid n = 0, 1, \ldots, N-1\}$, where symbol s_n generated on each iteration is defined as

$$s_n = \begin{cases} 0 & x_n \leq x_c \\ 1 & x_n > x_c \end{cases}. \quad (2.13)$$

The symbolic sequence $S = s_0, s_1, s_2, \ldots, s_n, \ldots, s_{N-1}$ is therefore a time series of 0s and 1s that aids in understanding which side of the state space I, the iterate x_n has visited. Hence, for the desired application, the signal that is intended to be measured can be input to the TM as initial condition x_0, and N-bit long symbolic sequence S can be generated with $T^{N-1}(x_0)$ iterations.

2.5.3 Symbolic Representation of the State Space

The critical point x_c is treated as a primary partition over the state space I that is sometimes referred to as Markovian partition [42] which restricts the two unique characteristics, defined on the two sides of a unimodal 1D map. For the TM, the primary partition divides the state space I into two halves or subintervals, [0,0.5] $\in I$ and (0.5,1] $\in I$ that respectively experience stretching and folding due to the map operation [13], [16].

As the $T(x)$ is operated over the entire state space, more partitions appear at the pre-images of x_c. Considering a current state, the pre-images [13] are the possible previous states that result into the current state on one operation of the map. Since TM performs two different operations on the either side of the critical point, every outcome of the TM iteration has two possible preimages, e.g. for a current iterate 0.5, there are 0.25 and 0.75 as pre-images, as, $2\mu(0.25) = 0.5$ also, $2\mu(1 - 0.75) = 0.5$, with ideally $\mu = 1$. For every operation of the map two more partitions are created within the subintervals about each of the previous

partitions. Therefore, for every n^{th} iteration, $(2^{n+1} - 1)$ partitions are generated and the state space I is divided into 2^{n+1} mutually exclusive sub-intervals I_j^n, where $j = 0, 1, 2, \ldots, (2^{n+1}-1)$, is the count of the sub-interval counting from the left boundary 0 to the right boundary 1 of I [13], [26]. In Fig. 2.16, it can be observed that for $n = 1$ iterations $(2^{1+1} - 1) = 3$ partitions have been generated that created $2^{1+1} = 4$ subintervals.

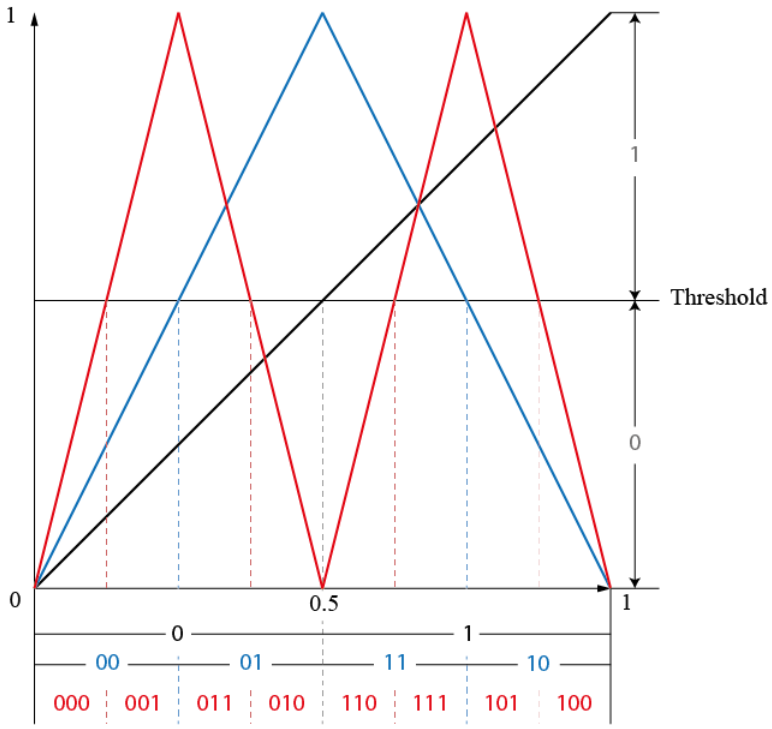


Fig. 2.16 Symbolic correspondence to the intervals of the state space

Any x input to the function originating from the state space, therefore, must belong to any *one* of the sub-intervals. Belonging to any interval on the either sides of any n^{th} level partition, a unique symbolic signature S with n+1 symbols can be generated when the input is iterated for n times. The symbolic states of the sequence are represented by the partitions generated at the n^{th} iteration, and can be used to identify or backtrack which of the I_j^n subintervals the input has originated from [13]. Therefore, an input or a dynamic state can be treated as either a point or an interval that can be defined by the corresponding symbolic

signature. The following properties establish the relationship between symbolic sequence S to the sub-intervals I_j^n generated by the map.

1. Every $x \in I_j^n$ result into the same symbolic sequence S up to n+1 iterations
2. If two initial conditions with the following identities $x \in I_j^n$ and $\breve{x} \in I_{j+1}^n$, then S and $\breve{S}$ differ by only one bit
3. $I_0^n \cup I_1^n \cup I_2^n \cup \ldots \cup I_{2^{n+1}-1}^n = I$
4. $I_j^n \cap I_k^n = \emptyset$ for $j \neq k$

From the properties 1, 2 and 4, it can be understood that an N-bit long unique symbolic identity S corresponds to a sub-interval of the size I_j^N and therefore, the longer S sequence will be (for higher N), the narrower will be the size of the I_j^N intervals. The S sequences corresponding to each j^{th} interval $I_j^N \in I$ can be formed into an ordered set, as shown in [13], [26], with an order of j = 0, 1, 2, …, 2^N relating with the number of I_j^N intervals that can be formed for N-bit long sequence as the way I_j^N are ordered in I. An example of such ordering of S consisting N = 3 bits with the corresponding j^{th} order of I_j^3, can be seen from Table 2.1.

Table 2.1 Correspondence between intervals and symbolic sequence

j	$\boldsymbol{S_3}(T,x)$	Binary	Intervals
0	000	000	0
1	001	001	0.125
2	011	010	0.25
3	010	011	0.375
4	110	100	0.5
5	111	101	0.625
6	101	110	0.75
7	100	111	0.875

Also, the partitioning with the symbolic codes corresponding to the intervals can be observed in Fig. 2.16. From such correspondence, each j^{th} interval can be denoted by the corresponding symbolic sequence S, hence, any initial condition x_0 with symbolic sequence S, the originating interval can be denoted by S as a subscript as in I_S^N, and it can be said that $x_0 \in I_S^N$ [13].

The originating interval can be determined by the process of tracing back the subintervals that are formed within intervals according to the symbolic sequence. The n^{th} symbol s_n in S, that is 1-bit of the sequence indicates whether x_n iterate belongs to the left or right side about the x_c, i.e., to the intervals I_0^0 or I_1^0. Hence, for a sequence S corresponding to $x_0 \in I_S^n$, with any $s_n \in \{0,1\}$ in S, $T^n(x_0) = x_n \in I_{s_n}^0$. Applying inverse operation would result into $x_0 \in T^{-n}(I_{s_n}^0)$. Therefore, to determine the originating interval of $x_0 \in I_S^N$ that satisfies all the N inverses through the entire symbolic path of the N-bit sequence, the inverse relation for every s_n is combined and the originating interval I_S^N can be defined as [13], [28]

$$I_S^N \equiv \bigcap_{n=0}^{N-1} T^{-n}(I_{s_n}^0). \tag{2.14}$$

To provide an example, a sequence S = 110...s_n is considered, the s_0 indicates that, the $x_0 \in I_1^0$. After applying TM once, the iterate $T(x_0) \in I_1^0$, therefore, the $x_0 \in T^{-1}(I_1^0)$. Again, for s_0, s_1, the $x_0 \in I_1^0 \cap T^{-1}(I_1^0) \equiv I_{11}^1 \subset I_1^0$ [13]. Thus, following all the symbols in the sequence in this manner, the originating sub-interval can be identified as

$$x_0 \in I_1^0 \cap T^{-1}\left(I_1^0 \cap T^{-1}(I_0^0 \ldots)\right) \equiv \ldots \subset I_{110}^2 \subset I_{11}^1 \subset I_1^0. \tag{2.15}$$

The inverse of map $T^{-1}(I_{s_n+1}^0)$ is applied depending on the s_n symbol. Since stretching and folding operations are performed on the two sides of the TM, the

transformation $T(x_n) = x_{n+1}$ has two possible preimages, meaning every x_{n+1} state has two possible x_n inverses. The correct inverse for the n+1th state is therefore identified through the symbolic path by looking at the previous n^{th} symbol whether s_n is 0 or 1 [13], with the following

$$x_n = T^{-1}(x_{n+1}) = \begin{cases} \frac{x_{n+1}}{2\mu} & s_n = 0 \\ 1 - \frac{x_{n+1}}{2\mu} & s_n = 1 \end{cases}. \quad (2.16)$$

Due to the negative slope on the (0.5,1] half of the map, the symbolic sequence generated, results in Gray codes which exhibit a mirroring effect on the sequence every time the critical point x_c is crossed. Since every such crossing is represented by '1', the stretching and folding behaviour of the map can be tracked by the count of 1's through the symbolic footprint of the trajectory. The odd count of '1' represents the folding operation and the even count represents the stretching operation [13]. It can be noted from Fig. 2.16 that the subintervals partitioned on the folding side I_1^0 = (0.5,1] are the mirror images of the subintervals on the stretching side I_1^0 = [0,0.5]. Due to the negative slope of the map on the folding side, the corresponding codes of subintervals become flipped and hence the symbolic coding of the intervals follows a fractal structure, exhibited by the Gray code, as can be seen in Fig. 2.17. A fractal can be defined

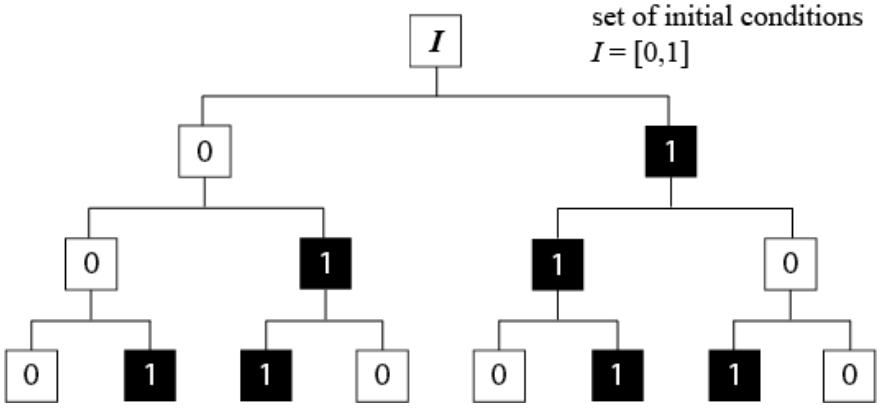


Fig. 2.17 Fractal structure of symbolic codes

as a recursively generated structure that exhibits self-similarity at all scales through repeating a fundamental graphical pattern [13], [43].

Any unimodal 1D map e.g. LM, TM can generate Gray codes, whereas the BM being defined by two stretching sides about the critical point, the symbolic code produced by BM follows the structure of binary codes, where one shift in the bit sequence depicts one iteration of BM being performed, which justifies the name of the map. Since S is a Gray code generated by the TM that correspond to the initial condition x_0, the symbolic code can therefore be realised and mapped back to the initial condition by converting the $S = s_0, s_1, s_2, \ldots s_n \ldots s_{N-1}$ to binary code $B = b_0, b_1, b_2, \ldots b_n, \ldots b_{N-1}$ using (2.17) and then further converting B to the real valued number using equation (2.18). The real valued representation of the Gray code is also referred to as Gray Ordering Number (GON), which can be used to order the S corresponding to different initial conditions within the state space by an order of the magnitude [26]. The following step is performed to convert an S to binary sequence B

$$b_n = \begin{cases} s_n & n = 0 \\ b_{n-1} \oplus s_n & n > 0 \end{cases}. \tag{2.17}$$

where, $\oplus$ is the Exclusive OR (XOR) logical operator. The B is further converted to GON through the following transformation (code in Appendix 2.8)

$$\mathrm{GON}(S) = \sum_{n=0}^{N-1} b_n^{-(n+1)}. \tag{2.18}$$

The GON conversion from S is however only valid and corresponds to the x_0 when the TM used to generate the symbolic signature is of full height, that is when the parameter is ideally $\mu = 1$. In practical implementations of TM, the

parameter is not always achieved to the ideal value, due to the operation of such a reduced height map the size of the subintervals and the corresponding structure of the symbolic code is altered. In the work contributed by Basu et al. [24], the problems of the altered subintervals have been addressed and therefore a different symbolic conversion scheme has been proposed accordingly.

2.6 Symbolic Shifting Window

In the symbolic dynamics, the shifting window [43] is a valuable tool to realise the correspondence between symbolic trajectories and the real iterates. Since it is already established that a symbolic sequence has a direct correspondence to the initial condition of a trajectory and given the fact that any iterate in a trajectory can be treated as an initial condition for the next iterates, a symbolic window of finite length can be shifted over the entire sequence to realise the possible symbolic correspondence to each of the real iterates in a trajectory. The mechanism of shifting window is described as follows.

For the trajectory $x_{N\text{-}1} = T^{N\text{-}1}(x_0)$ with the corresponding symbolic sequence S of length N, a symbolic window of size w bits can be shifted from $n = 0, 1, \dots, (N - w + 1)$, hence the sequence contained within the window can be defined as $S_n = s_{n+0}, s_{n+1}, s_{n+2} \dots s_{n+i} \dots s_{n+w\text{-}1}$. Assuming a shift operator ψ, operated on the S with a shifting window size w bit, one shift is therefore defined as $S_{n+1} = s_{n+1+0}, s_{n+1+1}, \dots s_{n+1+i} \dots s_{n+1+w\text{-}1} = \psi(s_{n+0}, s_{n+1}, \dots s_{n+i} \dots s_{n+w\text{-}1}) = \psi(S_n)$.

Given that the length of the original S sequence is N the total number of shifts can be performed with a window is $W = N - w + 1$. Also, GON for each S_n code from the shift can be calculated by converting S_n code to corresponding binary code $B_n = b_{n+0}, b_{n+1}, b_{n+2} \dots b_{n+i} \dots b_{n+w\text{-}1}$, as given by

$$b_i = \begin{cases} s_i & i = n \\ b_{i-1} \oplus s_i & i > n \end{cases}. \tag{2.19}$$

Accordingly, GON for each S_n is calculated as

$$\mathrm{GON}_n(S_n) = \sum_{i=n}^{N+w-1} b_i^{-(i-n+1)}. \tag{2.20}$$

In Fig. 2.18 the illustration of the discussed shifting window mechanism has been provided and how GON_n for each shifted window code is obtained has been shown.

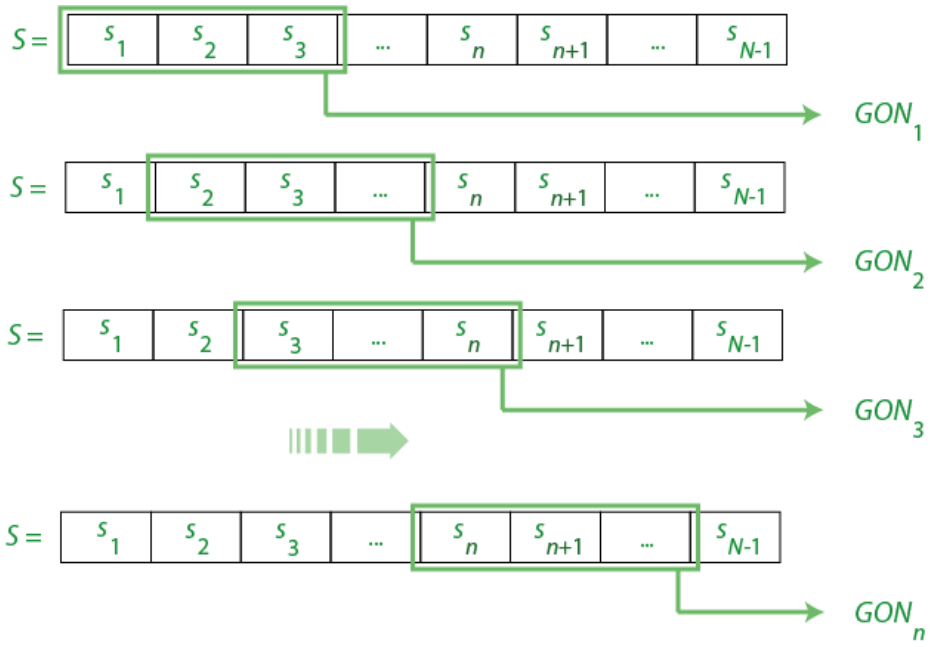


Fig. 2.18 Shifting window mechanism and obtaining GON

In Fig. 2.19 it can be visualised graphically that for an ideal parametric condition (μ = 1) of TM, the GON_n obtained from a shifting window over S has close resemblance to the actual real trajectory $x_{N-1} = T^{N-1}(x_0)$. The program for shifting window can be seen from Appendix 2.9. The accuracy of GON_n will depend on the adequate length w of S_n, and to match the trajectory up to sufficient length, higher N may be chosen.

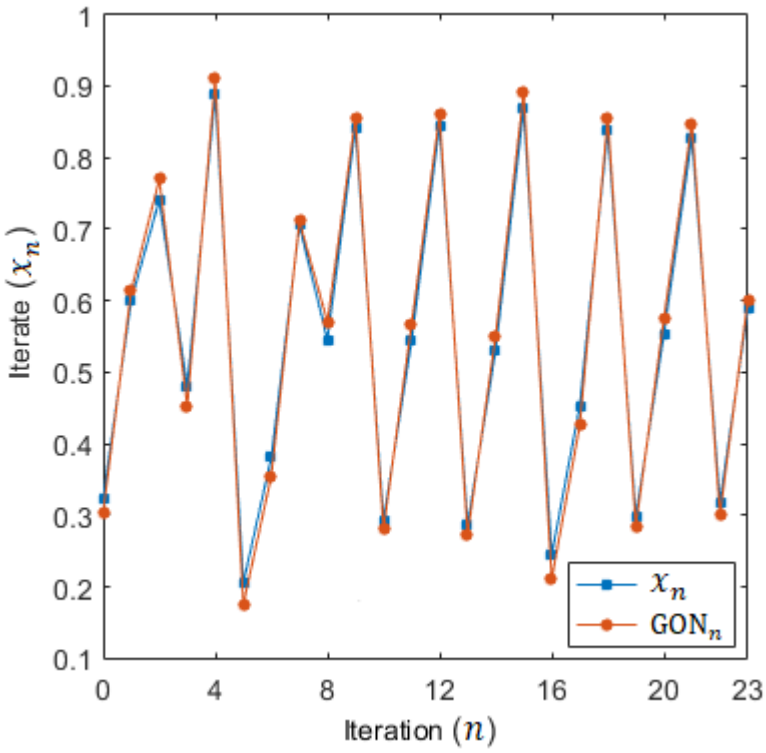


Fig. 2.19 Trajectories recreated through GON of shifting window

2.7 Kneading Theory: Symbolic Maximum and Minimum

The value of μ can be determined from the available symbolic dynamics, through realisation of the symbolic signatures corresponding to the point T_{max} or T_{min}. The symbolic correspondence of the dynamical maximum and minimum of the unimodal maps was proposed by Milnor-Thurston Kneading Theory, according to which: when a point equal to the value of x_c is input to a unimodal map (TM, LM), the outcome of the first iteration is equal to the T_{max} and the corresponding symbolic sequence S up to N iterations is known as the Kneading Sequence $\mathcal{K}$ [12], [13], [41] and thus can be expressed as

$$\mathcal{K} = S{:}(T^n(x_c)) = s{:}(T^0(x_c))s{:}(T^1(x_c))s{:}(T^2(x_c))\ldots s{:}(T^n(x_c)),\ n \in \mathbb{N}_0. \qquad (2.21)$$

$\mathcal{K}$ is an efficient tool to identify the symbolic sequences corresponding to T_{min} and T_{max} from the set of possible sequences that are generated by the

dynamics of a unimodal map at any parametric condition. Considering the shift operator ψ such that $s(T^{n+1}(x_0)) = \psi(s(T^n(x_0)))$, operations of ψ over $\mathcal{K}$ yield the sequences $S_{max} = \psi(\mathcal{K})$ and $S_{min} = \psi(\psi(\mathcal{K}))$, which correspond respectively to the dynamic maximum (2.10) and minimum (2.11) [13].

Therefore, if S_{min} or S_{max} can be realised from the generated symbolic dynamics, the information of the parameter μ is also recoverable. For any trajectory of x_0, a certain number of $\beta \in \mathbb{N}$ transient iterations can be chosen such that after β iterations, for any $n > \beta$, the iterates $x_n \in [T_{min},T_{max}]$ and the trajectory become bounded within the I'. The selection of β is empirical, depending on both the initial condition and the parameter of the map. Therefore, when both the factors remain to be unknown, β is chosen to be sufficiently large to ensure that the subsequent iterates of the trajectory belong within $[T_{min},T_{max}]$. For any initial condition originating outside the $I' = [T_{min},T_{max}]$ such that $x_0 < T_{min}$, then, after a count of β iterations until $x_{\beta+1} \geq T_{min}$, the symbolic sequence S: $T^{\beta}(x_0)$ will be a string of β zeros followed by a sequence S: $T^n(x_\beta) \in [S_{min}, S_{max}]$. For any initial condition $x_0 > T_{max}$, $x_0 \in [x_c,1]$, $s_0 = 1$, the $T(x_0) < T_{min} \in [0,x_c]$ and the dynamics will be continued according to the aforementioned behaviour. Such cases will have $s_0 = 1$ leading a string of β–1 zeros followed by a sequence S: $T^n(x_\beta) \in [S_{min}, S_{max}]$. From a sufficiently long trajectory, when the β transient symbols are discarded, and a shifting window is operated over the remaining sequence, the corresponding GONs of the sequences collected from each operation of the window can be ordered and hence can be matched to the ordering of S_{min} through S_{max}.

$$\mathrm{GON}_{min} \prec \ldots \prec \mathrm{GON}_{max} \equiv S_{min} \prec \ldots \prec S_{max}, \tag{2.22}$$

where, GON_{min} and GON_{max} are the respective GONs of S_{min} and S_{max}. Hence, once β symbols are discarded from a trajectory of any $T(x_0)$, theoretically, in the remaining $S_{N-\beta+1}$ part of the sequence S, there will be no symbolic sequence appearing in following dynamics for a given μ whose corresponding GON can be found outside the range $[GON_{min},GON_{max}]$. Therefore, once the dynamics enters the boundaries $[T_{min},T_{max}]$ any such sequences belonging outside $[S_{min},S_{max}]$ are not generated by the map and are treated as *forbidden sequences* [44] while all the sequences belonging within $[S_{min},S_{max}]$ are termed as *allowed sequence*. It is therefore confirmed that discarding the transient β symbols from a sufficiently long symbolic trajectory S and then operating symbolic shifting window over the sequence will let one to search for S_{min} or S_{max}, which in turn aid in determining the reduced parameter μ [24], [43], [45]. However, to determine the initial condition x_0 successfully from the corresponding S, it is not recommended to discard the transient β symbols because sufficient amount of information regarding the dynamic footprint of the originating point x_0 is contained in it. The originating interval can therefore be determined by back tracking every single symbolic iterate 0's and 1's in the available symbolic sequence. The, β transient symbols need to be discarded only when determining μ, and is kept intact while determining the initial condition x_0.

3 CHAOS BASED SIGNAL MEASUREMENT

The fundamental use of chaotic systems for signal measurement was contributed by Kennedy [10], where unimodal chaotic maps have been used as quantizers for ADC and measurement applications. Later more approaches [17]-[19], [24] have been proposed in this direction, which confirmed that measurement of a quantity using chaotic dynamics is possible, and there are vast scopes and possibilities for the development of a chaos based ADC.

While implementing a chaos function there are, however, a few issues that must be resolved before a standalone ADC can be achieved. The majority of the issues revolve around the physical implementation of the chaotic circuit, as the ideal dynamics of a chaotic function is greatly affected by several non-idealities caused in the electronic circuit. So far there have been several attempts to address some of the issues through both hardware and software approaches [18], [24], [43], [45], however there are plenty of scopes for future development as each of the newly proposed approach posed newer challenges that demanded further study and investigation.

3.1 Basic Challenges

Chaotic maps are simple to implement physically in the electronic hardware without engaging much of the resources. Therefore, physical implementation of chaotic functions has widely been proposed and achieved for various areas of applications. However, as discussed earlier, chaotic functions are very sensitive to electrical tolerances of the components used for the design, as the physical properties of materials used in electrical and electronic components are subject to

changes under several conditions e.g. temperature, electromagnetic interferences etc. Considering the inevitability of the non-idealities that are caused due to the tolerances in the components, the errors in the behaviour are often addressed through system level corrections rather than trying to engage highly accurate and precise components.

The non-idealities and errors in the implemented chaotic function are generally related to accuracy and precision errors that are caused due to the offset, temperature drift and noise depending on several physical and environmental factors. One of such non-idealities is the parametric reduction of the chaotic function where the map height is reduced due to parametric deviation, thus, affecting the dynamical behaviour of the map [43]-[45].

The other common type of non-ideality in the chaotic circuits is that the actual dynamical trajectories are altered due to the inherent noise in the system [46]-[48]. The noise adds random variables to each state of the dynamics thus the resultant evolution deviates from the ideal one and it becomes challenging to extract meaningful information out of such noisy trajectories.

3.1.1 Parametric Reduction of the TM

The ideal dynamics of the map for an initial condition is altered when the parameter is reduced, as can be seen in Fig. 3.1, where the dynamical trajectories for both ideal and non-ideal parameter conditions have been shown. Due to the parameter reduction, the partitioning of the intervals is asymmetric (not in equal halves as shown in Fig. 3.2) therefore the I_S^N subintervals appear in uneven sizes causing the initial points to be redistributed unevenly to the adjacent intervals within state space *I*.

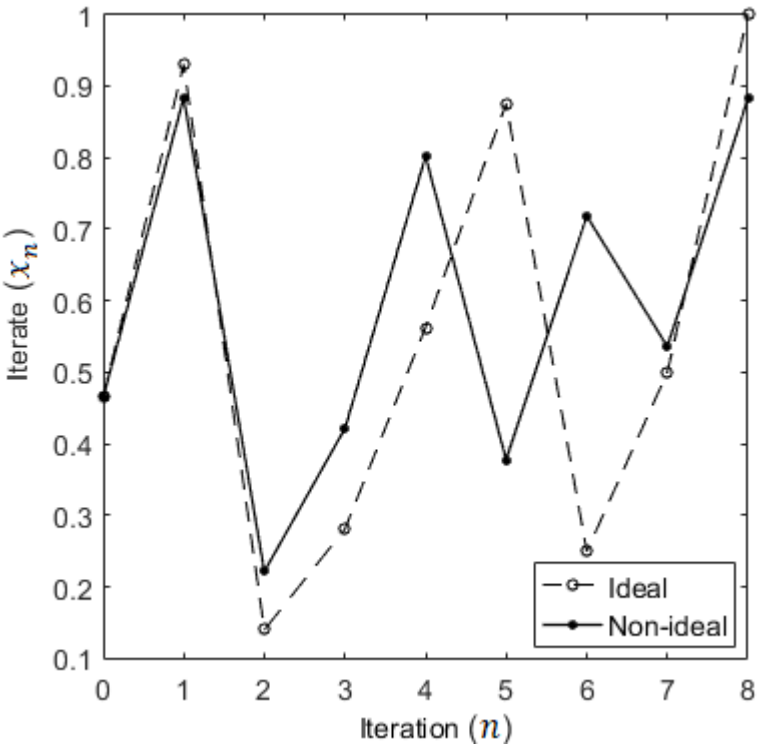


Fig. 3.1 Altered dynamics due to reduction in parameter value

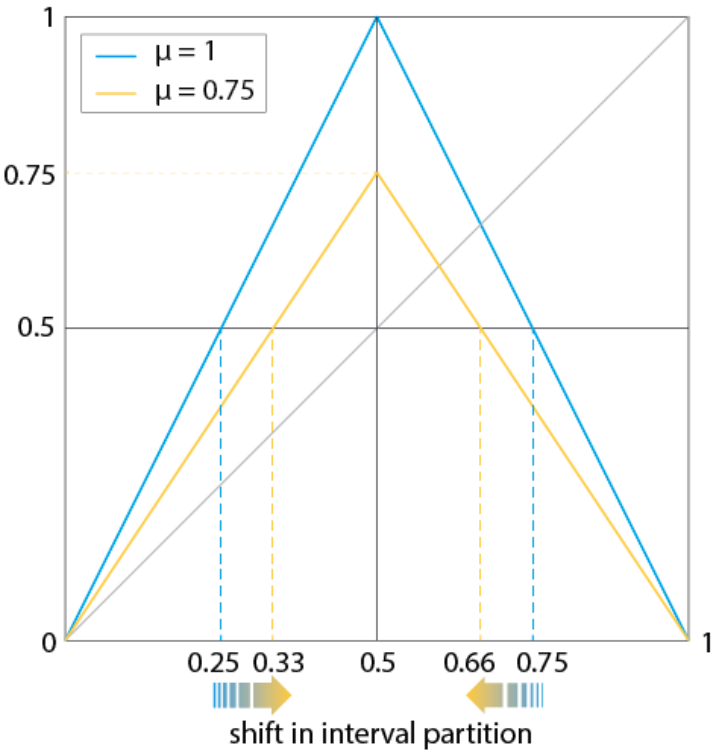


Fig. 3.2 Partitions get shifted generating asymmetrical intervals

The farther the parameter is deviated away from the ideal value, the greater will be the amount of shift in the partitions from the ideal positions [24]. Therefore, the correspondence between the symbolic codes S and the intervals also become altered as the codes are unevenly mapped in the state space I.

In Table 3.1 symbolic correspondence to the initial condition for both ideal and non-ideal parameters have been shown, and the GON of the symbolic sequences have been calculated. It can be noticed that the GONs for the non-ideal cases have greatly deviated from the ideal values for a slight change in parameter.

Table 3.1 Sequences generated with $\mu = 1$and $\mu < 1$

x_0	S_{16} for $\mu = 1$	S_{16} for $\mu < 1$	GON for $\mu < 1$
0. 1951	0010100100001011	0011101101001110	0.17790
0. 1952	0010100100000100	0011101101001000	0.17796
0. 1953	0010100100000000	0011101101011010	0.17802
0. 1954	0010101100000111	0011101101011100	0.17808
0. 1955	0010101100001010	0011101101010110	0.17814

3.1.2 Impact on Initial Condition Estimation

There have been several contributions in the field that proposed techniques to estimate initial condition similar to that of calculating GONs. These approaches primarily convert Gray codes to the real values from the equivalent binary codes by applying base 2 algebra, as described in equations (2.17) and (2.18). Such approaches will not return accurate result for practical situations where the parameter value is not ideal. Given that the partitions are misplaced, and symbolic trajectories deviate from the ideal, when sequences are converted into GON values, there can be found a loss of correspondence with the actual initial condition. Similar observations have been documented by Litovski et al. in [18], where directly converting the Gray codes generated by a reduced height TM to

the real valued signals led to incorrect mapping. The initial condition estimation problem was further investigated in [27] where it was concluded that for a correct conversion and mapping, ideal map parameter is necessary, which is however, quite challenging to achieve with physical electronic circuitry. Kapitaniak et al. had also conducted a broad study in [17] towards measurement of electrical signals using symbolic dynamics of chaotic maps. Their observations confirmed that the signals estimated from the symbolic dynamics of the electronically implemented TM were greatly affected by the offsets and tolerances of the components used which significantly reduced the parametric height of the map. In Fig. 3.3 the conversion of symbolic sequences following the conventional or GON approach has been shown for a ramp of initial condition where the symbolic sequences generated through reduced parameter TM.

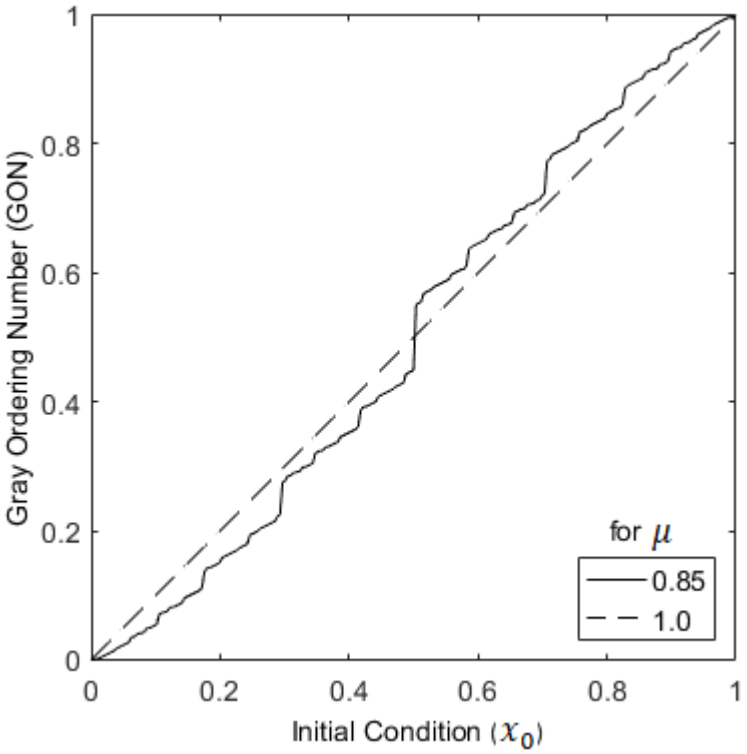


Fig. 3.3 GON estimation with reduced parameter

It can be observed that a change in parameter to $\mu = 0.85$ has caused a substantial deviation in the estimates from the actual values. The GON transformations have limitations towards exactly estimating the initial conditions from the symbolic sequence generated through a map with non-ideal parameter, as such transformation apparently assumed that the intervals are partitioned into equal halves even in the case of reduced height map. However, GON transformations are still useful for the relative ordering of the symbols in the state space as symbolic sequences can be assigned a real valued magnitude for comparison within a range.

Further investigations have been conducted by Cong et al. [28] who established theoretically that for any chaotic map, initial condition can be estimated from the symbolic sequences with reasonable accuracy by applying reverse map from the last symbol of the symbolic sequence. The method assumed the real valued iterate for the last symbol is either $x_{N-1} = 0.5$ for $s_{N-1} = 0$ or $x_{N-1} = 1$ for $s_{N-1} = 1$, then according to the symbolic path, the reverse map is applied as described in equation (2.16). Such process might add an overhead time for recording the data before the conversion can begin, hence it might affect the conversion speed. In [24], Basu et al. developed a method based on interval theory that took the partitions and uneven intervals formed due to the non-ideal parameter into consideration. The method established an approach to estimate the initial condition from the first symbol instead of the last one relying on the process of forming subintervals within intervals as the dynamics evolve, thus the conversion can be run as a pipelined process in parallel with the symbol acquisition, and the conversion overhead time can be reduced. However, for both the proposed advancements, the availability of the knowledge of the non-ideal parameter is

necessary that must be supplied for the fruitful estimation. Also, noise is another non-ideality that affects the dynamical evolution. Several noise reduction methods have been studied and proposed in the past that also rely on a fair amount of knowledge about the system producing the dynamics. For the complete knowledge of the dynamical system, knowledge of the control parameter is therefore essential to define the source system. In the following sections, the way in which the dynamics is affected by noise has been described, and the proposed solutions for the noise corrections have been elaborated.

3.1.3 Dynamical Behaviour Affected by Noise

Due to the stretching and folding nature of the TM, the state space is partitioned on every iteration and points in the invariant interval $I \subset \mathbb{R}$ *eventually* map densely over $I \subset \mathbb{R}$, [13], [41], which leads to any point $x_0 \in I$ generate unique trajectories that hold the key information of the initial condition. The iterative trajectories can be treated as a footprint of the system dynamics and its initial condition.

When chaotic maps are implemented in physical hardware, e.g. electronic circuits, the actual dynamics of $\mathcal{X}$ trajectories, governed by the feedback process, become greatly affected by the inherent noise in the system hardware. Fig. 3.4 shows the feedback mechanisms of the TM dynamics and how it can get affected by noise in the iteration process. Noise affected chaotic trajectories have two constituent parts, one of which is the deterministic part, that is governed by predefined set of rules related to the chaotic function. The other is the random or indeterministic part, which is introduced by noise that is intrinsic to the physical system.

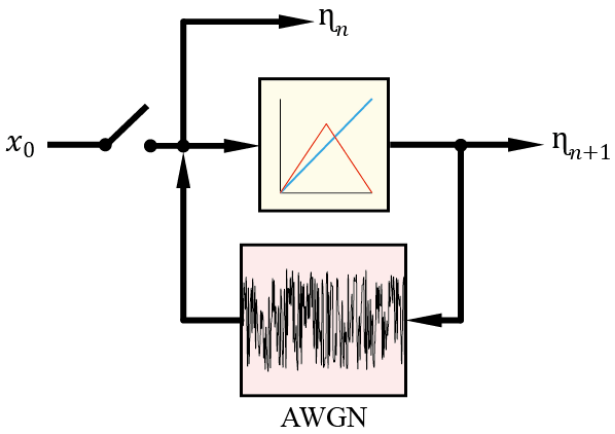


Fig. 3.4 Noise affected TM iterations in a feedback mode

The coexistence of noise in the feedback process of every iteration affects the dynamics by adding random variables to every iterative state. Such types of noise can be described as dynamical noise [46], and its evolutionary process can be defined as

$$\eta_{n+1} = T(\eta_n) + \xi_n, \tag{3.1}$$

where, η_n is the n^{th} noisy iterate and ξ_n is the random variable combined at each stage of iteration.

The behaviour of the random variables ξ_n can be realised by the properties of additive white Gaussian noise (AWGN) due to its intrinsically additive random variables showing Gaussian distribution [48]. The random distribution of AWGN has a *zero mean* whose variance is characterised by the signal-to-noise-ratio (SNR) with $10\log_{10}(\sigma_x^2/\sigma_\xi^2)$ measured in dB, where σ_x and σ_ξ are the standard deviations of the signal and the noise respectively. In electronic hardware, the thermal noise is best represented by AWGN. The AWGN has been simulated in the programs using the built-in function *awgn*(x_n,SNR) of MATLAB, where, the input arguments are the state variable x_n and the SNR.

Let the trajectory affected by the dynamical noise be defined as $\eta = \eta_0, \eta_1, \eta_2, \ldots, \eta_{N-1}$. The initial condition x_0 is the original signal entering from an independent source which can be assumed to be not affected by dynamical noise of the chaotic system yet, such that $\eta_0 = x_0$. However, as the dynamical evolution continues the noise is also propagated through the iterative process (as can be seen in Fig. 3.5), hence affecting the original trajectories of the initial conditions. It is therefore, a challenge to extract meaningful information from the trajectories corrupted by dynamical noise.

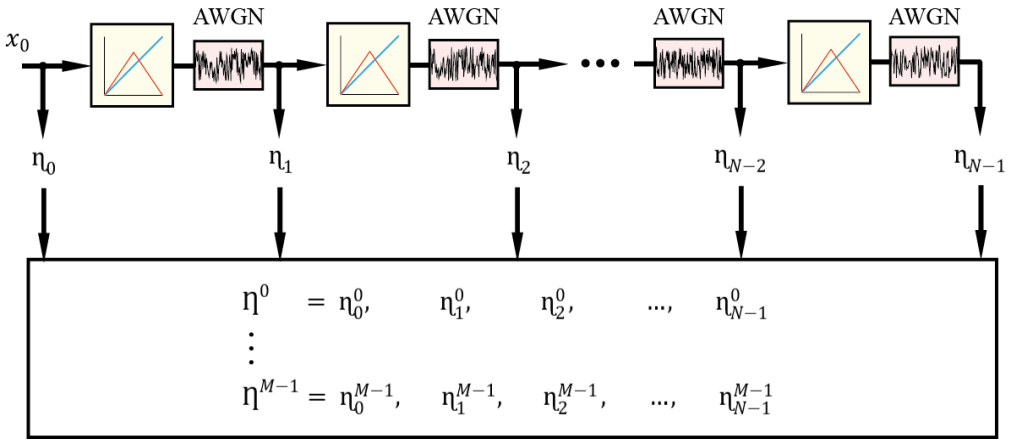


Fig. 3.5 Dynamics of TM affected by noise in every stage of iteration

The behaviour of the dynamical noise can be studied by sampling a single trajectory repeatedly, and then by observing the samples collectively. For a collective view, M samples of η^m for an initial condition x_0 can be collected. Each of the sampled trajectories can therefore be represented as

$$\eta^m = \eta_0^m, \eta_1^m, \eta_2^m, \ldots, \eta_{N-1}^m, \tag{3.2}$$

where $m = 0, 1, \ldots, M-1$. For any m^{th} trajectory, each n^{th} iterate is sampled and stored for all the N iterations in the trajectory and then for the next $m{+}1^{\text{th}}$ trajectory, the sampling process is repeated again from η_0^{m+1} through to η_{N-1}^{m+1} as

illustrated in Fig. 3.5. Hence, for any n^{th} iterate, there will be M samples of the noisy data available for observation.

Sensitive dependence on initial conditions has a key role to play in dynamics, since little perturbations in every iterative stage lead the trajectories to different paths. For any perturbation p_n, around the close neighbourhood of the actual iterate x_n, the resulting $x_{n+1} + \mathrm{p}_{n+1} = T(x_n + \mathrm{p}_n)$ is further deviated compared to the original transformation $x_{n+1} = T(x_n)$, as established by rate of divergence or the Lyapunov exponent, given by equation (2.5). As long as the $|\mathrm{p}_{n+1}| > |\mathrm{p}_n|$ or the Lyapunov exponent λ is positive, the trajectories of TM will be divergent in nature [12]. Similar divergence is also experienced by the dynamics when such perturbations as r_n is randomly introduced (replacing p_n by r_n in equation 2.5) in every stage of iteration.

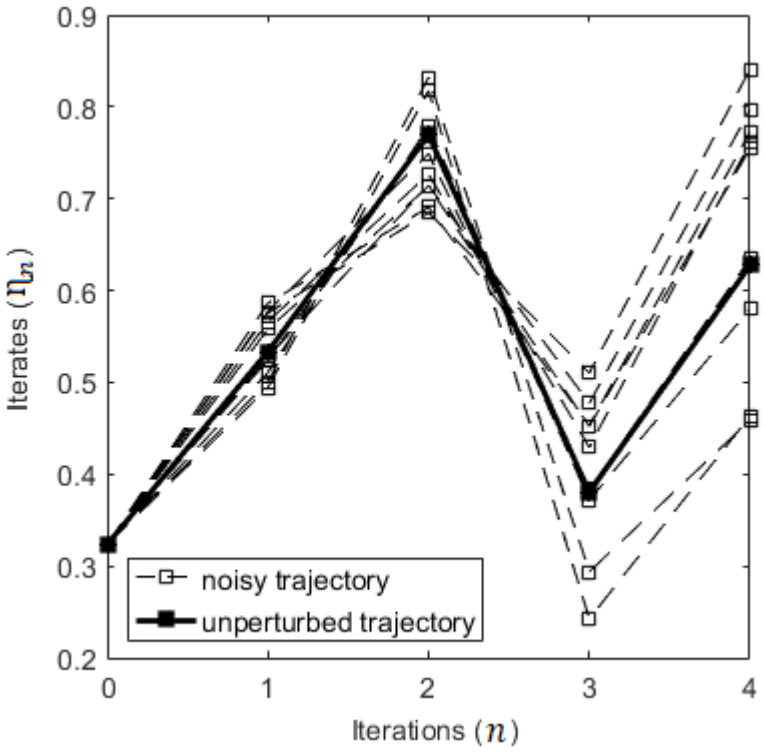


Fig. 3.6 Divergent noisy trajectories of an initial condition

A diverging behaviour of a trajectory can be seen in Fig. 3.6 where a short length (N = 5) trajectory of an arbitrary initial condition x_0 = 0.3234 through TM with parameter μ = 0.825 is perturbed by noise with SNR = 30 dB and a few (M = 10) samples are collected for observation.

Due to the stretching and folding nature of the TM, there is a dense mixing of the trajectories in the entire state space [13]. Hence, the points that are originating from a close neighbourhood will be eventually spread all over the state space $I = [0,1]$, as it may be the case with η_n^m points that are separated by minute perturbations due to the noise. When the dynamics is continued for a higher N, and a large set of M number of trajectories are observed collectively, the η^m are found to be highly distributed over I. In Fig. 3.7 the η^m trajectories for, x_0 = 0.3234, μ = 0.825, N = 10, SNR = 30 dB and M = 50 are shown.

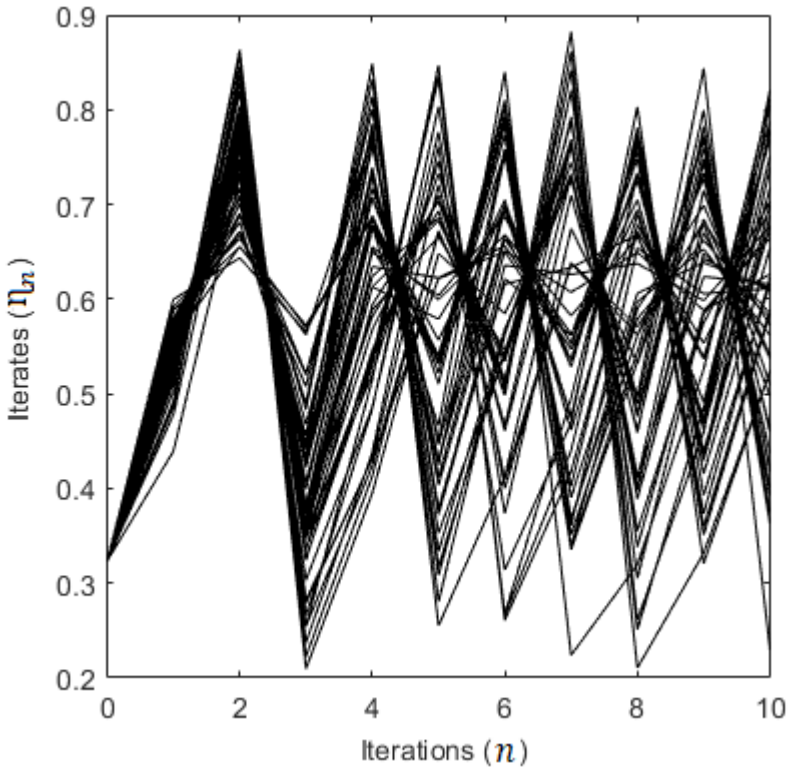
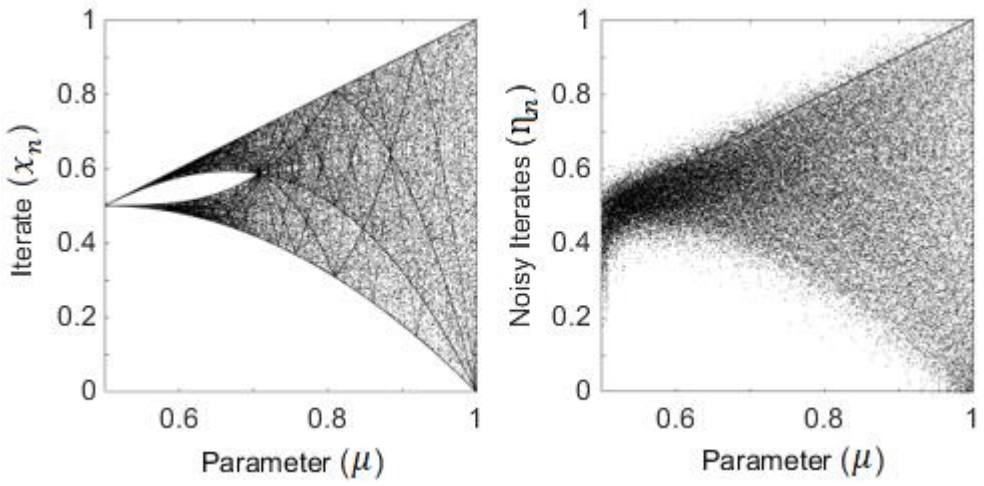


Fig. 3.7 Noisy trajectories with SNR = 30 dB

It is often the case with implemented maps that the information regarding the parameter μ is not known and is required to be extracted from the only other available information, i.e. the collected set of trajectories. In a noise-free system, μ may be extracted from the relationship between any known pair of x_n and x_{n+1} in a trajectory. However, for noisy iterates, since $\eta_{n+1} = T(\eta_n) + \xi_n$ and ξ_n cannot be exactly quantised, such calculations will not determine the correct μ [29]. Although it may seem intuitive to consider searching for the map maximum T_{max} in order to pursue μ, however, even for signals with noise levels as low as SNR = 30 dB, the noise causes the iterates to exceed T_{max}, therefore causing the map maximum to be lost. This is easily verified through the bifurcation diagram of a noise affected tent map in Fig. 3.8 (a), as compared to noise-free distribution of the dynamics in Fig. 3.8 (b).



(a) (b)

Fig. 3.8 Bifurcation diagrams of noise-free and noise affected tent map

Given that the TM being the chosen chaotic function for the purpose of signal measurement, identification of the control parameter of the map offers two-fold advantage, that are: using the information of the parameter, less-noisy data can be

retrieved from the noise affected time series; also, the initial condition can be determined utilising the knowledge of the parameter.

As described in [49]-[51], to determine the actual signal from the noisy data, an efficient noise reduction method must be able to determine or approximate the underlying chaotic function. Since, both chaotic trajectories and noisy samples are wideband signals, linear filtering techniques (e.g. lowpass, bandpass filters) or classical Fourier approaches cannot be adapted to reduce noise, because deterministic chaotic trajectories might also be discarded by the filtration [52], [53]. The works proposed in [54]-[56] offer efficient solutions to significantly reduce noise involving reconstruction of the dynamical phase space. The methods highlighted the importance of knowledge of the system or the approximation of the source function for which, a knowledge of the system parameter is essential. It is then possible to approximately identify the deterministic and indeterministic parts from the available data. To properly identify the chaotic function responsible for a certain nature of dynamics, identification of the control parameter is necessary. Once the function under operation and the control parameters are known, then the dynamics produced by such a function can be completely discerned from noise. A property of the tent map has been observed, where the dynamical noise can be utilised to determine the control parameter of the map. Since noise allows the dynamics to spread in the entire state space, the properties seen to be statistically prevalent in the state space. In the following chapters, it is shown how the sampled noisy trajectories η^m can be utilised to determine the value of μ, and establishment of the property and techniques to apply it in finding the parameter has also been described.

3.2 Parameter Estimation: Current Techniques & Limitations

There have been quite a few contributions that approach parameter estimation of chaotic maps, through symbolic time series or the real iterates, depending on the requirements of the application. In this section the most salient of all the contributions and approaches that showed the key direction towards solving the parameter estimation problem will be discussed. In [22], the trajectories generated by the unimodal chaotic maps were first analysed in terms of symbolic patterns represented by L and R respectively depicting left and right side of the map. The work proposed an idea of ordering the symbolic patterns by estimating a numerical value corresponding to each of the unique symbolic patterns. It was fundamentally shown that a set of patterns would occur in the symbolic trajectories – for the dynamics originating from the midpoint $C = 1/2$ and again coming back to the same point after a few iterations, such patterns up to a certain length are unique for a given parameter of the map. Each of the different patterns for a certain length N of the sequence were ordered by forming an equation with the midpoint of the map. Examples of all the patterns that are possible for length $N = 5$ for a unimodal transformation $f_\lambda^N(x)$, with originating point as C and again returning back to C on the 6^{th} time step is shown as the following:

$C \to R \to L \to R \to R \to C$

$C \to R \to L \to L \to R \to C$

$C \to R \to L \to L \to L \to C$

It is to determine for which parameter λ the transformation $f_\lambda^N(x)$ would map from C to C in $N = 5$ steps. Therefore, the following equation holds:

$f_\lambda^N(½) = ½.$ (3.3)

Solving the equation for λ generated the order number which directly corresponded with the map parameter. Therefore, such patterns can be used for ordering of the sequences generated by a map and hence the parameter value of the map can be determined from the symbolic analysis. During that period the method showed a new direction to analyse chaotic dynamical trajectories using symbolic identities and patterns. However, it was difficult to order the patterns as unique combinations by looking at the shorter sequences of *L* and *R*, and therefore longer observation was necessary – sometimes in millions of iterates – to ensure that a large set of unique patterns have been gathered for the analysis.

From the fundamentals established by Stein et al, later in this direction the idea of Kneading sequence has formed. Further developments have been contributed by Wu et al. in [43], who made some propositions towards analysing the Kneading sequence in terms of Gray codes and established the properties of the Kneading sequences in terms of symbolic patterns of S_{min} and S_{max}. They have also proposed that to estimate the map parameter, the Kneading sequence of a unimodal chaotic map can be generated by iterating the map with an initial condition $x_0 = 0.5$ i.e. the value of the critical point. Alternatively, the Kneading sequence can be searched over the long-term symbolic trajectories. As an approach to determine the parameter from the Kneading sequence, once the S_{max} is found and GON(S_{max}) is determined, since GON(S_{max}) $\neq \mu$, they proposed a search algorithm for the parameter between two test parameter boundaries. The test parameter boundaries with lower and upper bounds are defined as p_L and p_U respectively. In the processing domain where the parameter is being estimated,

the map $T(x_c)$ is operated separately with test parameter p_T = (p_L+ p_U)/2 and the symbolic sequence S is obtained, and following conditions are applied to narrow down the test parameter boundary, that are: if GON(S) > GON(S_{max}), then, p_U = p_T, or, if GON(S) < GON(S_{max}), then, p_L = p_T, and the map is iterated with new test parameters p_T = (p_L+ p_U)/2 until, GON(S) = GON(S_{max}), then, p_T is the desired solution. The method involves several indefinite search steps to converge to actual solution within the test boundaries, also for each new test parameter, the map is separately iterated in the processing domain and GONs are compared, this might cause a problem in determining the parameter from the perspective of numerical processing as the estimator might have to wait indefinitely for the solution to converge. The estimation of the parameter from the Kneading sequence is however, not straight forward; further developments in this direction have been contributed later in Chapter 3.

A different approach for parameter estimation of unimodal maps has been proposed by Alvarez et al. in [45], which involved finding the probabilities of all possible order patterns that can be generated for a given parameter of the map. The technique was formulated by obtaining a long-term symbolic trajectory for a given parameter condition from the iterative dynamics of the map. Then small symbolic patterns were extracted from the symbolic trajectory by operating a shifting window, followed by taking permutations of the extracted patterns which would result in creating all possible patterns up to a certain length that the map can generate for a parameter. The number of patterns is counted and compared with the patterns that can be obtained for the full (ideal) parameter of the map. In reduced parameter conditions, patterns that can be generated through the map, do not cover universally all possible patterns, as all possible patterns can only be

realised when the map parameter is full. Therefore, the probability, determined by a ratio between the number of patterns that can be realised through the dynamics or by the permutations, and the number of all possible patterns in the symbolic dynamics, will correspond to the map parameter. As the map parameter is improves, the probability of the possible patterns would improve – tending to be 1. On the context of TM, the issues regarding such a probabilistic approach is: to find all possible patterns an enormously long dynamical trajectory is needed, and a large number of permutations needed for each extracted code to cover all possible patterns that a map can generate. Usually, such parameter estimation methods are applied in the area of communication and encryption where sequences are generated through a map implemented in digital computing domain therefore generating the dynamics for iterations more than thousands is not a problem. However, for hardware-oriented applications as signal measurement, suitable techniques need to be investigated.

4 PARAMETER ESTIMATION METHODS

In this chapter, the methods to estimate the parameter of the TM have been proposed. From the knowledge of the dynamics of the TM and understanding of the anomalies that can be caused by parametric non-idealities, two independent methods have been explored during the course of investigation of the problem. Following are the detailed description of the methods in the form of algorithms.

4.1 Parameter Estimation: Kneading Sequence Approach

It has been observed that the dynamics of the TM shows dense distribution over the state space I. For parameter $\mu \in (0.5,1]$ there exist unique maximum and minimum points. The map maximum $T_{max} = T(x_c) = \mu$ correspond to the parameter value, therefore for a non-ideal condition of μ, if the maximum point T_{max} can be determined from the available dynamics, the parameter value is recovered for the reduced height map. It is understood that gathering a set of as many points mapped by the dynamics, will ensure that the entire distribution of points can define an interval I' with boundaries, T_{max} and T_{min}, and a search for T_{max} can be performed within I'.

A similar search can be performed in symbolic space to determine the sequence that corresponds to the T_{max}. The method in this work is based on Kneading sequence, which is an improvisation of the approach proposed in [24]. Since a real valued dynamical iterate can be represented by a symbolic identity, a shifting window of finite length can be operated over the entire symbolic dynamical trajectory to retrieve the sequences that correspond to each of the real iterates. Similarly, for the case of symbolic search, it is recommended that sequences with

as many iterates as possible are gathered, to accommodate as many shifts as possible. Once all the sequences corresponding to the shifts of the symbolic shifting window are extracted, the gathered sequences can be ordered by the determining GON of each sequence. Hence the sequence corresponding the maximum can be determined and the value of μ can be estimated from that sequence.

4.1.1 Proposed Kneading Sequence Search Algorithm

From the properties of Kneading sequence, it is understood that if a long-term dynamical trajectory is gathered for an arbitrary initial condition, it is highly probable that the dynamics will reach the maximum value $S_{max} = \psi(\mathcal{K})$ at some point in time confirmed by the fact that chaotic dynamics is highly distributed over the state space. A search technique can therefore be devised to determine the S_{max} and the Kneading sequence $\mathcal{K}$ corresponding to the operating map maximum from a symbolic trajectory S. A "Binary Search" approach to identify S_{max} has been proposed; utilising a shifting window operated over a symbolic sequence of x_0. As discussed in Section 2.9, in order to avoid the initial transient β symbols before the dynamics can enter I' and correspondingly in the domain $[S_{min}, S_{max}]$, initial β bits of symbols from the symbolic sequence S need to be discarded. Since the β is empirically chosen, it needs to be sufficiently large, such that the monotonic transitions towards $[S_{min}, S_{max}]$ are discarded.

The search algorithm for S_{max} is given as:

1. A sufficiently large sequence S is obtained for any arbitrary initial condition x_0. S is stored in the digital domain for further processing

2. The $0 < \beta \in \mathbb{N}$ transient symbols must be discarded from S. After discarding β symbols, the remaining sequence $s_\beta, s_{\beta+1}, \dots, s_n, \dots, s_{N-1}$ is of size $N-\beta+1$ bits. The index of n will now be considered from $n = \beta, \beta+1, \beta+2, \dots, N-1$

3. A finite length symbolic window of the size w bits is shifted over sequence $s_\beta, s_{\beta+1}, \dots, s_n, \dots, s_{N-1}$. The symbolic window is shifted towards right by one bit, such that,

$$s_{\beta+1}, s_{\beta+2}, \dots s_{\beta+i+1}, \dots s_{\beta+w} = \psi(s_{\beta+0}, s_{\beta+1}, \dots s_{\beta+i}, \dots s_{\beta+w-1}), \quad (4.1)$$

where each shift is operated by ψ, the total number of shifts can be performed $W = N - \beta - w + 1$. The index $i = 0, 1, 2, \dots, w-1$ defines position of each bit inside the symbolic window.

In Fig. 4.1, the shifting window approach has been illustrated with further detail of how GON is estimated from the w-bit symbols appearing each shift of the window.

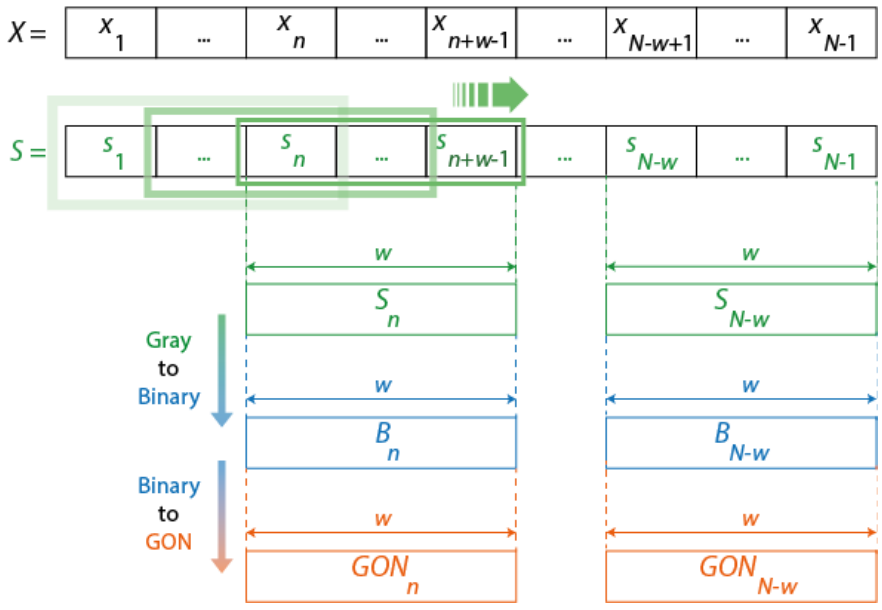


Fig. 4.1 Operation of shifting window and determining GON of each shift

4. The real valued GON from w bit long sequence appearing in the window on each shift is obtained. The sequence with symbols $s_0, s_1, \dots, s_{w-1}$ in the window is converted to equivalent binary $b_0, b_1, \dots, b_{w-1}$ through

$$b_i = \begin{cases} s_i & i = 0 \\ b_{i-1} \oplus s_i & i > 0 \end{cases}, \quad (4.2)$$

where b_i is the i th digit of the binary code. Further GON is estimated as

$$\text{GON} = \sum_{i=0}^{i=w-1} b_i \,.\, 2^{-(i+1)}. \quad (4.3)$$

5. For each shift of the symbolic window over S for $n = \beta, \dots, N-1$, GON_n are estimated and compared with GON_{max}. If $\text{GON}_n > \text{GON}_{max}$, then the previous GON_{max} can be overwritten as $\text{GON}_{max} = \text{GON}_n$, since the process initially started with $\text{GON}_{max} = 0$, on every n^{th} shift, GON_n is calculated and GON_{max} is updated if the stated condition is satisfied. This process is repeated until the last shift, the stored largest GON value is then found as GON_{max} and the corresponding sequence of GON_{max} can be recorded and referred to as the maximum sequence S_{max}.
6. The S_{max} is confirmed to be the sequence representing $T_{max} = T(x_c)$. According to the explanation in Section 2.9, since the x_c is represented by the Kneading sequence $\mathcal{K} = S{:}(T^n(x_c))$, the first symbol of $\mathcal{K}$ is 0, and as $S_{max} = \psi(\mathcal{K})$, to obtain the Kneading sequence $\mathcal{K}$ from S_{max} a 0 must be appended in front of the S_{max}.

7. Due to the non-ideal parameter $\mu < 1$, $\text{GON}(\mathcal{K}) \neq x_c$. The deviation of $\text{GON}(\mathcal{K})$ from x_c can be realised by calculating the difference $(2\mu)^{-i}$-2^{-i} at each i^{th} stage of symbolic conversion. Since binary-to-decimal conversion is performed by choosing a base of 2, where the sequence through TM is generated by non-ideal μ, the difference may be realised in terms of 2-2μ.

Combining the differences for all i stages would result into the total difference by which GON($\mathcal{K}$) is away from x_c.

8. As $\mathcal{K}$ is a Gray code, there is certain rule that needs to be followed in order to combine the differences. As described in Section 2.6, for every '1' appearing in the sequence, even count of 1's refers to stretching nature (positive slope) and odd count of 1's refers to the folding nature (negative slope) of the TM operated on an interval. Therefore, the signs for each i^{th} stage of the differences are accordingly adjusted to compensate for the deviation of the actual signal value from the corresponding GON. After appending '0' in front of $S_{max,}$ the size of $\mathcal{K}$ is w+1, the following rule is applied to obtain the differences relative to symbolic order in the sequence $\mathcal{K}$:

    a. Starting from the MSB (most significant bit) of the sequence $\mathcal{K}$, i.e., s_0, the number of 1's appearing in the sequence is counted as γ

    $$\gamma_i = \gamma_{i-1} + s_i. \tag{4.4}$$

    b. The difference for each i^{th} stage according to odd or even nature of γ_i is calculated as δ_i

    $$\delta_i = \begin{cases} s_i\left(2\mu^{-i} - 2^{-i}\right) & \gamma \; is \; odd \\ -s_i\left(2\mu^{-i} - 2^{-i}\right) & \gamma \; is \; even \end{cases}. \tag{4.5}$$

    The alternating 1's, i.e. whether γ_i is odd or even, would decide whether the corresponding difference will be added or subtracted (as per the positive and negative slope of the map). The symbols s_i, whether a 0 or a 1 in each i^{th} stage is multiplied with the

difference, which decides whether or not any difference will be added or subtracted at a particular i^{th} stage, as for s_i = 0 no difference is generated i.e. δ_i = 0, for s_i = 1, $\delta_i = \pm s_i\left(2\mu^{-i} - 2^{-i}\right)$

9. Once all the δ_i are determined from $\mathcal{K}$ in terms of $\pm s_i\left(2\mu^{-i} - 2^{-i}\right)$, the differences are added to the GON($\mathcal{K}$), and is equated to x_c in the following form

$$\delta_0 \pm \delta_1 \pm \delta_2 \ldots \pm \delta_{w+1} + \text{GON}(\mathcal{K}) = x_c. \quad (4.6)$$

Since x_c = 0.5 and the GON($\mathcal{K}$) is known the equation can be solved in terms of μ as each $\delta_i = \pm s_i\left(2\mu^{-i} - 2^{-i}\right)$ while the μ is unknown.

10. The above equation is a polynomial, hence the solution for μ will return multiple roots with few containing imaginary parts. The largest root, that is non-zero and non-negative with the imaginary part equal to 0 should be selected as the estimated μ.

The proposed method is straightforward that is to find S_{max} over a symbolic trajectory and determine the Kneading sequence $\mathcal{K}$ from S_{max}. The parameter value is determined by solving the difference equation derived using $\mathcal{K}$. The method can be easily implemented in the computing domain (MATLAB code using single input in Appendix 2.10).

4.2 Parameter Estimation from Noisy Dynamics of TM

The samples of noisy iterates can be used to determine the map parameter. Given that the information of both the time step (n) and the magnitude of each iterate (η_n^m) is contained in a trajectory, a sampled iterate can be treated as a point in a

two-dimensional cartesian coordinate system with *n* plotted in *X-axis* and the η_n^m plotted in *Y-axis*. When the consecutive iterates η_{n-1}^m and η_n^m in the η^m trajectories are joined together through straight lines, a set of intersections within such straight-lines appear as all the sampled trajectories are observed collectively [29], as can be seen in Fig. 3.7 in Section 3.1.3, when η^m trajectories are viewed through line-plots for all M. It can be noticed that, such intersections appear in a concentrated neighbourhood between the majority of the n^{th} and n+1$^{\text{th}}$. It can also be observed that these intersections mainly appear at about the same level on the *Y-axis* of the plot, the behaviour of these intersections is further studied. In the following experiment noisy trajectories are plotted for two different values of μ perturbed with same level of noise SNR = 25 dB in the system for an arbitrarily chosen initial condition x_0 = 0.3234 for both the experiments. In Fig. 4.2 it may be noticed that the clusters have appeared around the corresponding fixed point x_f = 0.6226 for μ = 0.825, marked with a dashed line.

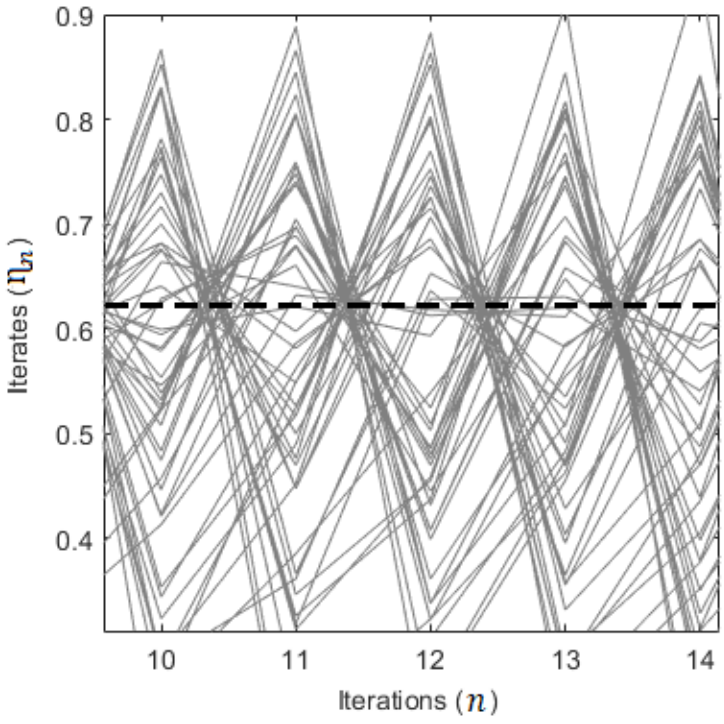


Fig. 4.2 Crossovers around x_f = 0.6226 for μ = 0.825

Similarly, in Fig. 4.3, the clusters marked with a dashed line have appeared around x_f = 0.5745 for μ = 0.625. Collectively the locations of the cluster of intersections on the *Y-axis* is relatable to the location of the non-zero fixed point x_f of the TM, as the clusters of intersections have appeared in different locations for different parameter values [29]. Hence, it is verified that such clusters have a correspondence with x_f and therefore the μ of the map can be determined if x_f is identified according to the property 6 in Section 2.5.

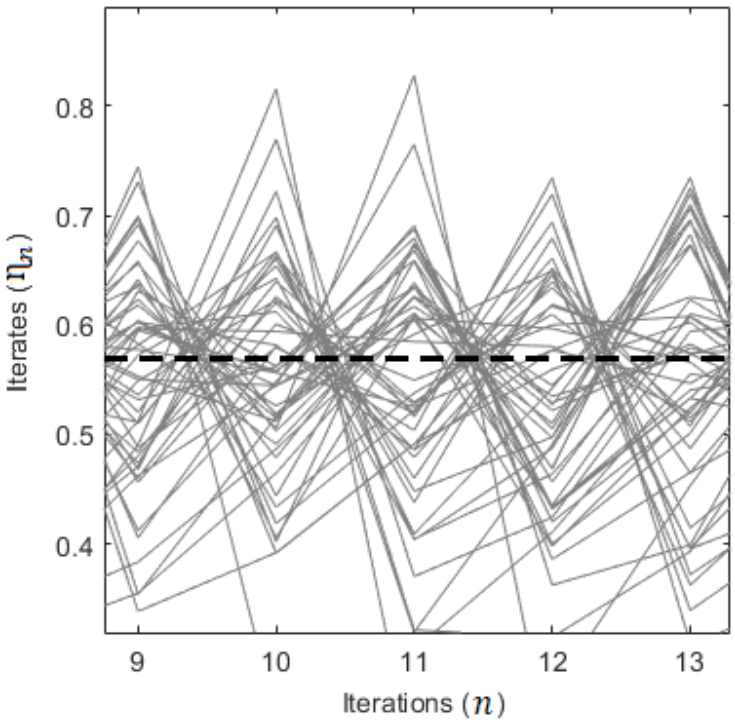


Fig. 4.3 Crossovers around x_f = 0.5745 for μ = 0.625

To ascertain this perspective, further study and exploration of the dynamic behaviour of the state space around the neighbourhood of the non-zero fixed point x_f is necessary. The mapping of the points within I is observed for a single iteration from which different intervals are identified that show unique mapping properties. For any parameter $\mu \in (0.5,1]$, the preimage of x_f is given by x_p =

$x_f/2\mu$. Therefore, x_{n+1} for any $x_n \in [0,x_p) \in I$ will be less than or equal to x_f, thus the mapping $T : [0,x_p) \mapsto [0,x_f)$ holds.

On the other hand, any $x_n \in [x_p,x_c) \in I$ the corresponding x_{n+1} will be greater than x_f, and the mapping will be $T : [x_p,x_c) \mapsto [x_f,\mu)$. Also, for the points within the intervals $[x_c,x_f) \in I$ and $[x_f,1] \in I$ show the respective mappings $T : [x_c,x_f) \mapsto [x_f,\mu)$ and $T : [x_f,1] \mapsto [0,x_f]$. If the x_n from the above intervals and the corresponding $x_{n+1} = T(x_n)$, is plotted on a two-dimensional coordinate system where the *X-axis* represents *n* and *n*+1, and the *Y-axis* represents x_n and x_{n+1}, then a straight line joining the two points can be constructed.

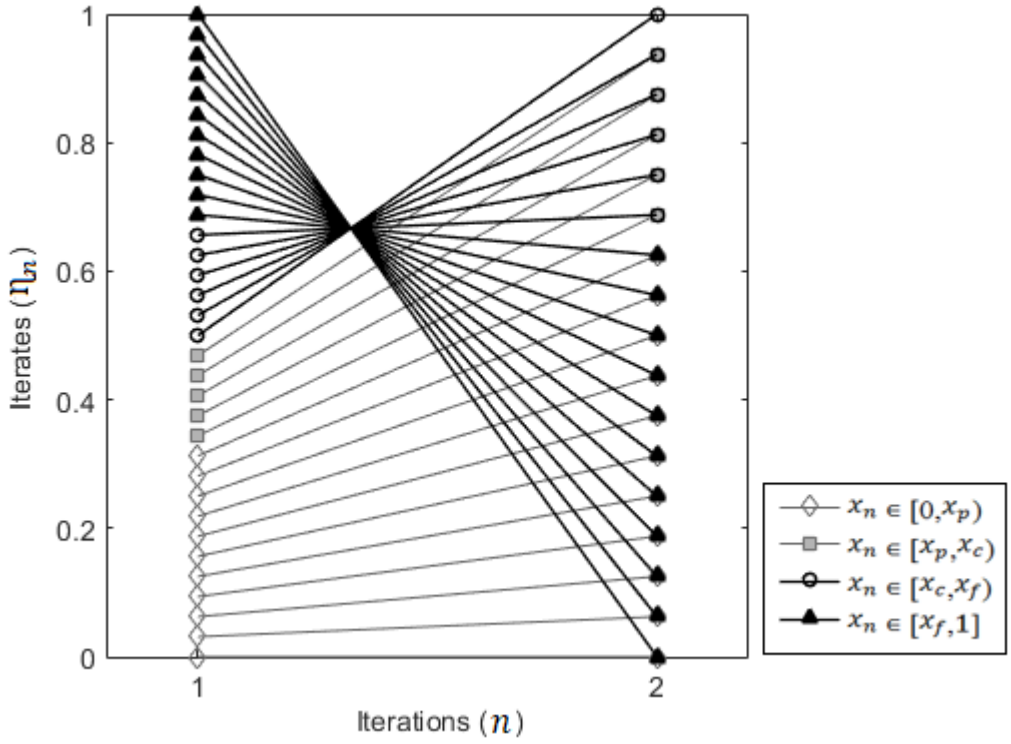


Fig. 4.4 Mapping within the state space

In Fig. 4.4 the lines joining the of x_n and the corresponding x_{n+1} for the intervals within $x_n \in [0,x_p)$, $x_n \in [x_p,x_c)$, $x_n \in [x_c,x_f)$, $x_n \in [x_f,1]$ have been shown. It is clearly seen that the straight lines formed by the iterates within the two intervals

$[x_c,x_f) \mapsto [x_f,\mu)$ and $T : [x_f,1] \mapsto [0,x_f]$ about the x_f, intersect at a single point x_f on the *Y-axis*. Whereas, for the remaining intervals, the intersections between the x_n and corresponding x_{n+1} points are not concentrated on a single point; rather, the intersections are spread widely over the *XY-plane*. Hence, it is confirmed that points within 50% of the entire state space I, i.e. $x_n \in [x_c,1] \in I$, will show such intersections at x_f [29].

Given that the state space is highly distributed due to the perturbed dynamics of the TM, having noise in the iterative process will have additional advantages by maximising the chances of the dynamics spreading over the entire state space and therefore the chance of finding the intersections concentrated around a single point x_f of the map is maximised. Thus, from the sampled collection of the noisy trajectories, locations of such intersections can be determined between the iterates, and can be further be correlated with x_f of the map to determine the parameter μ.

4.2.1 The Algorithm: Parameter Estimation from Noisy Trajectories

From the observations presented in the previous section, it is understood that, from a collection of noisy trajectories, the intersections corresponding to the non-zero fixed point (x_f) of the map can be determined closely, that can be utilised to identify the map parameter with a reasonable accuracy.

A statistical approach has been implemented to estimate the fixed point from the collection of η^m trajectories, as given in the following algorithm.

1. The N number of η_n^m iterates are collected for each m^{th} sample trajectory η^m.
2. According to the behaviour discussed in the previous section, to find the intersections that appear closely around x_f, the criteria $x_n \in [x_c,1] \in I$ needs to be fulfilled. Therefore, for an n^{th} set of sampled iterates the η_n^m points that satisfy $\eta_n^m \in [x_c,1]$ should be selected. Between any given n^{th} and $(n+1)^{\text{th}}$ iterates, let the total number of selected points out of M samples be $M' \leq M$ contained in set $H_n = \{\eta_n^m \in [x_c,1]\}$. In Fig. 4.5 the selection of the noisy samples has been illustrated.

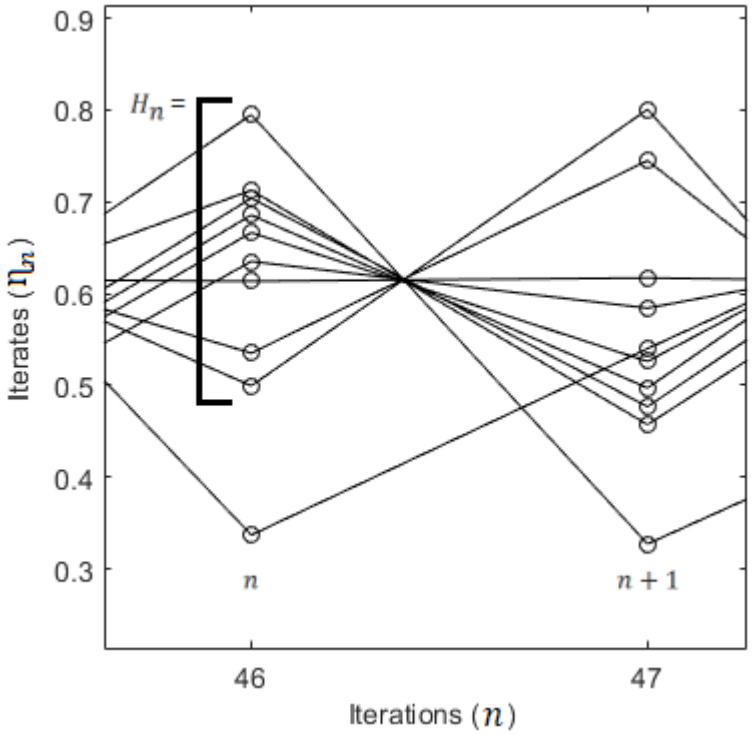


Fig. 4.5 Selection of the iterates to determine the intersections

3. M' number of straight-lines have been formed with each element in H_n with their corresponding $(n+1)^{\text{th}}$ iterates.
4. The number of intersections between straight-lines formed by the unique pairs of points η_n^i, η_{n+1}^i and η_n^j, η_{n+1}^j for the M' selected cases will be

$M'(M' - 1)/2$, for all $i = 0, 1, \ldots, M' - 1$ and $j = i + 1, i + 2, \ldots, M' - 1$ such that $i \neq j$.

5. The ordinate value Y^k of the intersection is solved by equation (4.7) in terms of η_n^i, η_{n+1}^i and η_n^j, η_{n+1}^j

$$\frac{Y^k - \eta_n^i}{\eta_{n+1}^i - \eta_n^i} = \frac{Y^k - \eta_n^j}{\eta_{n+1}^j - \eta_n^j}, \tag{4.7}$$

where, k = 1, 2, …, $M'(M' - 1)/2$. In Fig. 4.6 the assignment of coordinates to the samples that are used to determine the intersections have been illustrated.

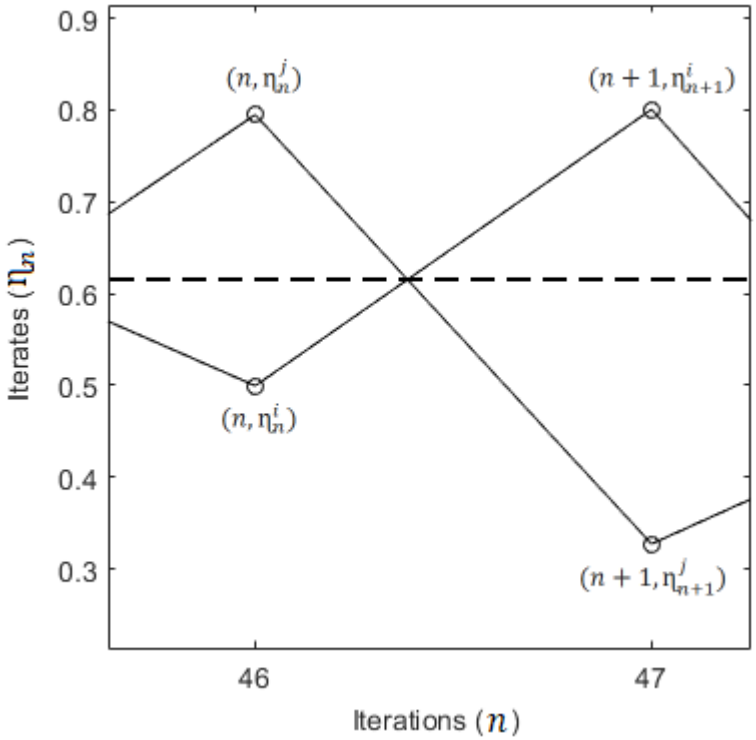


Fig. 4.6 Assignment of coordinates to the selected samples

6. The Y^k solutions (intersection) form a cluster of the points by the lines joining between n and $(n + 1)$ time steps. The central point within each such cluster is determined by calculating the *arithmetic mean* $\bar{Y}_n$ from all the Y^k solutions for a given n.

$$\bar{Y}_n = \frac{1}{M'(M'-1)/2} \sum_{k=1}^{M'(M'-1)/2} Y^k. \quad (4.8)$$

Note: The selection criterion $\eta_n^m \in [x_c, 1]$ for any n^{th} time step might lead to H_n being empty or singleton set (i.e. $|H_n| < 2$), which might generate no solution for Y^k and therefore $\bar{Y}_n$. To have at least one Y^k solution for an intersection between n and $(n+1)$, there must be at least two elements in H_n; therefore, any such $|H_n| < 2$ and the corresponding $\bar{Y}_n$ must be excluded, otherwise it might lead to undesirable outcomes in the programming domain.

7. Also, as between every n and $(n+1)$ time-steps there would be one $\bar{Y}_n$; then, the total number of $\bar{Y}_n$ produced for all $|H_n| \geq 2$ must be $\Theta \leq N - 1$. Hence, from the $\bar{Y}_n$ values again a single point ξ that is the closest approximation of x_f can be determined by calculating the arithmetic mean of Θ number of $\bar{Y}_n$. Using the value of ξ the control parameter of the TM can be estimated as μ' using the following equation given by re-writing equation (2.9) in terms of ξ and making μ' the subject

$$\mu' = \xi/2(1-\xi). \quad (4.9)$$

The technique shown in this section utilises the samples of the noisy iterates to determine the crossovers of the fixed point and the map parameter. The proposed algorithm can be implemented in the computing domain (code in Appendix 2.12) where the noisy iterates sampled from the physical hardware can be processed.

5 RESULTS: PARAMETER ESTIMATION

The two parameter estimation methods, as described in Chapter 4, are tested through simulation. The results of both the methods – described in Sections 4.1 and 4.2 respectively – are detailed in this chapter. First, the Kneading Sequence Search Algorithm is simulated and verified for performance. Next, the second method, Parameter Estimation from Noisy Dynamics of TM has been performed. A thorough analysis of the estimated outcomes and errors has been presented.

For the computerised simulations, MATLAB R2016b has been used. Alternatively, the open source Octave can also be used. However, for speedy convergence MATLAB is recommended, as it heavily utilises parallel processing. For the result generation and storage in the programs, one- and two-dimensional array structures have been extensively used. The programs for each of the algorithms have been developed considering that the methods can be implemented on digital computation devices (e.g. Microcontrollers, FPGAs) and can be adapted for programming languages such as C, C++, Java, VHDL etc. In the first part of every experiment (as in the programs in Appendices 2.10 – 2.13), the TM dynamics is operated with a test parameter and an initial condition, the time series trajectories were generated along with the symbolic outcomes for a certain length *N*. In case of noise-oriented approaches, *awgn*(x_n,SNR) function is used to perturb each x_n state with White Gaussian Noise corresponded by a chosen SNR and the trajectories were sampled for *M* times. In the later parts of the experiments, parameter estimation algorithms have been applied on the generated data set, and the estimation results and errors have been observed and plotted graphically as detailed in the following sections.

5.1 Results: Kneading Sequence Search Algorithm

The Kneading sequence search algorithm and parameter estimation technique from the Kneading sequence using difference equation approach has been evaluated for various parametric conditions. Following is an example to illustrate how the S_{max} and $\mathcal{K}$ sequences are determined by operating the symbolic shifting window over the S sequence; followed by that, the TM parameter value μ is estimated by solving the difference equation with x_c, realised according to the symbolic order of $\mathcal{K}$. An arbitrary initial condition $x_0 = 0.092904$ is iterated with a test parameter $\mu = 0.90$, for $N = 200$ times to ensure that the dynamics reach closest to the map maximum value within that many iterations.

The symbolic sequence S is obtained and the algorithm to find $\mathcal{K}$ is operated as follows:

1. Initial $\beta = 5$ transient symbols were discarded to avoid the monotonic trajectories that might not be a part of the dynamical attractor. As can be seen from Table 5.1, a symbolic window $w = 12$-bit wide is operated over the S, starting from $s_5, s_6, \ldots, s_{16}$ and the $GON_5 = 0.295410$ for the code within the symbolic window is estimated. The $GON_{max} = 0$, as initialised and compared with GON_5, if $GON_5 > GON_{max}$, then $GON_{max} = GON_5$.
2. As the single step shift is operated by ψ, in Table 5.1, it can be observed that the GON_6 for the new code appearing in shifted window is estimated as $GON_6 = 0.591064 > GON_{max} = 0.295410$, therefore $GON_{max} = GON_6$
3. On the next shift, $GON_7 = 0.817382 > GON_{max} = 0.591064$, therefore $GON_{max} = GON_7$.
4. $GON_8 = 0.364746 < GON_{max} = 0.817382$, so GON_{max} remains unchanged.

5. As the shifting of the symbolic window is continued the maximum GON was found to be GON_{63} = 0.923095 for the entire symbolic sequence (see Table 5.1).

Table 5.1 Shifting window of 12-bit operated over symbolic sequence

	β = 5					Symbolic Window, w = 12-bit													
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
S	0	0	0	1	1	0	1	1	0	1	1	1	0	0	1	1	1	1	
						GON_5 = 0.295410													
		Symbolic Window, w = 12-bit																	
n	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
S	0	1	1	0	1	1	1	0	0	1	1	1	1	0	1	0	0	1	
		GON_6 = 0.591064																	
			Symbolic Window, w = 12-bit																
n	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
S	0	1	1	0	1	1	1	0	0	1	1	1	1	0	1	0	0	1	
			GON_7 = 0.817382																
				Symbolic Window, w = 12-bit															
n	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
S	0	1	1	0	1	1	1	0	0	1	1	1	1	0	1	0	0	1	
				GON_8 = 0.364746															
	⋮																		
				Symbolic Window, w = 12-bit															
n	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	
S	1	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	0	0	
				GON_{63} = 0.923095															

6. The code within the symbolic window corresponding to GON_{63} has been recorded as S_{max}. Form S_{max} the $\mathcal{K}$ has been created by appending a 0 in front of the S_{max}, as given in Table 5.2.

Table 5.2 Obtaining Kneading sequence $\mathcal{K}$ from S_{max}

i	0	1	2	3	4	5	6	7	8	9	10	11	12
S_{max}	1	0	0	1	1	0	1	0	0	1	1	1	-
$\mathcal{K}$	0	1	0	0	1	1	0	1	0	0	1	1	1

7. The GON($\mathcal{K}$) = 0.461547 is determined which is not equal to the value of x_c. The difference between x_c and GON($\mathcal{K}$) is the factor of the non-ideal μ, which is realised from the order of the symbols s_i in $\mathcal{K}$ in terms of $(2\mu^{-i} - 2^{-i})$ by applying rule 8(a) and 8(b) in the proposed algorithm (see Section 4.1.1) and formed into an equation with x_c, as $\mathcal{K}$ corresponds to x_c. Solving the equation will determine the unknown value of μ.

Table 5.3 Kneading sequence

	MSB →												LSB
$\mathcal{K}$	0	1	0	0	1	1	0	1	0	0	1	1	1

Starting from the MSB of $\mathcal{K}$ (as given in Table 5.3), as per the proposed technique, the following rule is applied.

$$\gamma_i = \gamma_{i-1} + s_i. \tag{5.1}$$

$$\delta_i = \begin{cases} s_i\left(2\mu^{-i} - 2^{-i}\right) & \gamma\ is\ odd \\ -s_i\left(2\mu^{-i} - 2^{-i}\right) & \gamma\ is\ even \end{cases}. \tag{5.2}$$

a. Starting with i = 0, the γ_0 = 0, i.e. even as the s_0 = 0, so δ_0 = 0

b. For i = 1, the γ_1 = 1, i.e. odd as the s_1 = 1, so $\delta_1 = (2\mu^{-1} - 2^{-1})$

c. For i = 2, the γ_2 = 1, i.e. odd as the s_2 = 0, so δ_2 = 0

d. For i = 3, the γ_3 = 1, i.e. odd as the s_3 = 0, so δ_3 = 0

e. For i = 4, the γ_4 = 2, i.e. even as the s_4 = 1, so $\delta_4 = -(2\mu^{-1} - 2^{-1})$

8. After determining all the δ_i for all s_i in the $\mathcal{K}$. The following equation is obtained by adding all the differences to GON($\mathcal{K}$) and equating with x_c.

$\delta_1 - \delta_4 + \delta_5 - \delta_7 + \delta_{10} - \delta_{11} + \delta_{12}$ + GON($\mathcal{K}$) = x_c. (5.3)

$\Rightarrow \delta_1 - \delta_4 + \delta_5 - \delta_7 + \delta_{10} - \delta_{11} + \delta_{12} + \text{GON}(\mathcal{K}) - x_c = 0.$

Substituting the δ_i terms in the equation with $\left(2\mu^{-i} - 2^{-i}\right)$,

$\Rightarrow (2\mu^{-1} - 2^{-1}) - (2\mu^{-4} - 2^{-4}) + (2\mu^{-5} - 2^{-5}) - (2\mu^{-7} - 2^{-7}) + (2\mu^{-10} - 2^{-10}) - (2\mu^{-11} - 2^{-11}) + (2\mu^{-12} - 2^{-12}) + \text{GON}(\mathcal{K}) - x_c = 0.$

Putting the values GON($\mathcal{K}$) = 0.461547 and x_c = 0.5 in the equation the highest real valued solution of the μ is given as

$\Rightarrow \mu = 0.897065$

that is close to the actual parameter value μ = 0.90.

In Fig. 5.1, it can be observed that the δ_i differences cumulatively build up to x_c from GON($\mathcal{K}$). Hence, it is understood that the equation has been established appropriately utilising the structure of the symbolic code that can be solved in terms of the unknown parameter value.

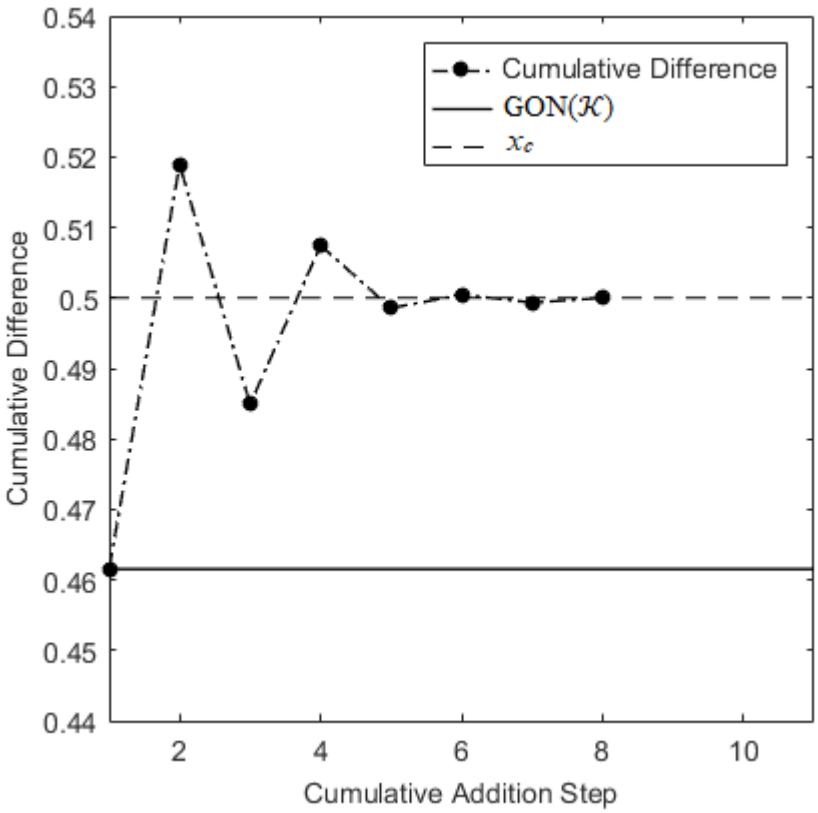


Fig. 5.1 The equation build-up: differences added to GON($\mathcal{K}$)

In the graph shown in Fig. 5.1, the differences added with GON($\mathcal{K}$) given in the equation (5.3) has been plotted in the following order GON($\mathcal{K}$) + δ_1, GON($\mathcal{K}$) + $\delta_1 - \delta_4$, GON($\mathcal{K}$) + $\delta_1 - \delta_4 + \delta_5$, GON($\mathcal{K}$) + $\delta_1 - \delta_4 + \delta_5 - \delta_7$, and so on in each step, as can be seen at the last step of addition the differences added with GON($\mathcal{K}$) equals to x_c = 0.5. In Fig. 5.2 the estimation results for the parameter values ranging from μ = [0.8,1] has been graphically analysed.

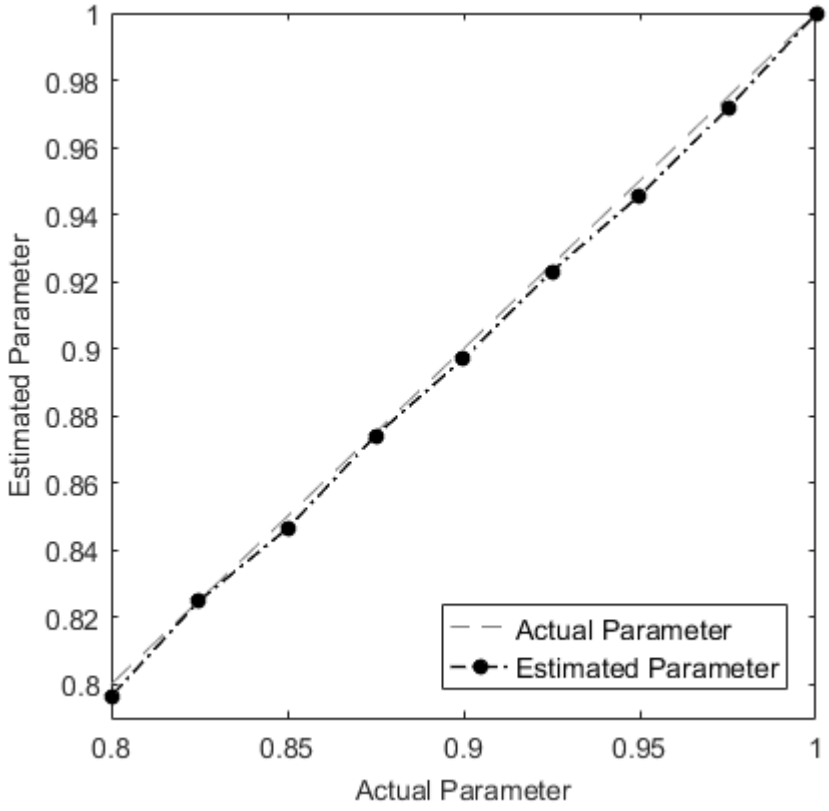


Fig. 5.2 Estimated parameter values for different parametric conditions

Symbolic sequences S were generated from the arbitrary initial conditions using the parameter values μ = [0.8,1]. To ensure that S_{max} appears in the dynamics, the length of S has been sufficiently chosen as N = 200, and a symbolic window of length w = 12 bits have been operated over S. Thus, $\mathcal{K}$ is determined from the S_{max}. It can be seen that the estimated results for different parameters are in good agreement with the actual parameter values, as can be further observed clearly

from Fig. 5.3 where the estimation error has been shown in percentages that is approximately below 0.5%.

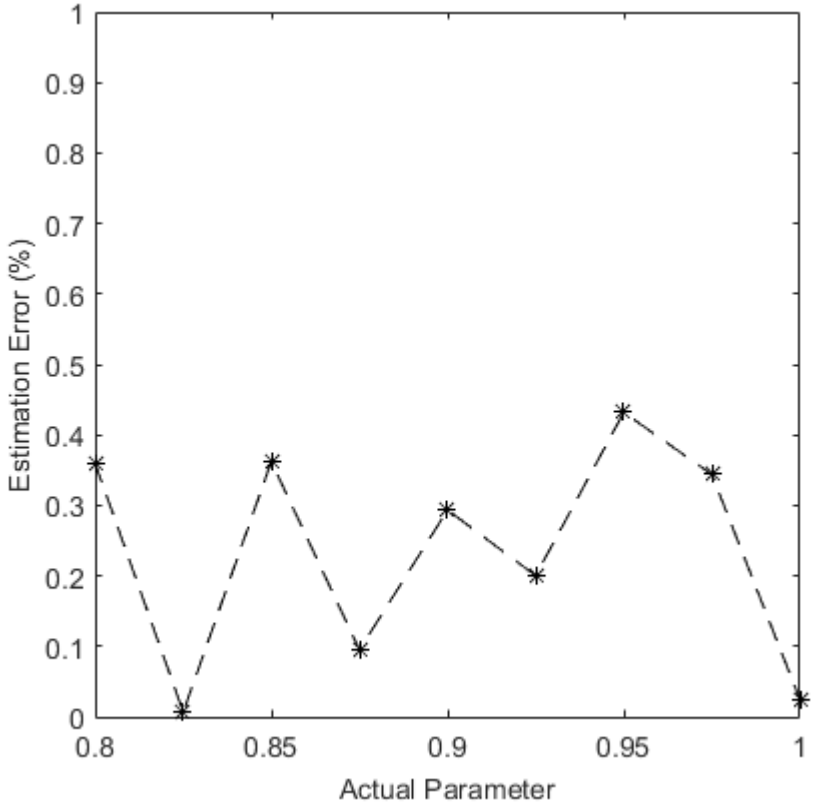


Fig. 5.3 Percentage error in parameter estimation

The parameter estimation algorithm has been evaluated for a set of initial conditions. As the dynamics for different initial condition and parameter values would map to the maximum T_{max} with the corresponding S_{max} appearing at different times in the time series trajectory, it can be ensured that for N = 200 after discarding β = 5 symbols, the parameter can be estimated with sufficient accuracy. A set of initial conditions has been chosen as x_0 = [0,1] within the state space with resolution $1/2^8$, that were iterated with parameter μ = 0.8, and parameter was estimated from the S generated with each different initial condition (code for the entire set of initial condition used to generate the results is included in Appendix 2.11).

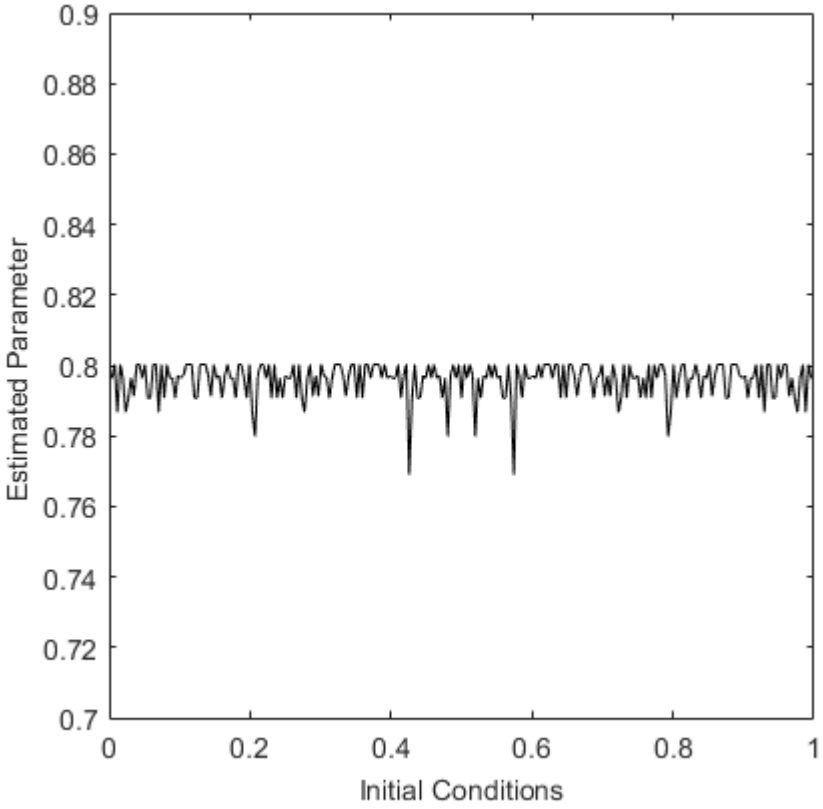


Fig. 5.4 Estimated parameter for all initial conditions

The results presented in Fig. 5.4, shows the majority of the estimated outcomes with reasonable accuracy belonging within range [0.78, 0.8] which is close to the actual parameter value $\mu = 0.8$.

For the effective results, the choice of certain variables involved in the algorithm, deserves some discussion. Though β is empirically chosen, the choice of β is only to ensure that the transient initial points of the dynamics before entering the attractor are discarded (elaborate details can also be found in Sections 2.5.1 and 2.7). Since the initial point or the input may appear from anywhere within the state space and not necessarily within the bounds $I' = [T_{max}, T_{min}]$, the initial few iterates may map to the points outside this attractor. Once the dynamics enters the attractor, it remains within, and therefore never exceeds the map maximum. In a bid to avoid misrepresentation of the map maximum, any transient iterates prior

to entering the attractor is therefore discarded. Hence the choice of β must be enough, so that it sufficiently ascertains that the dynamics has entered the attractor originating from any initial condition. The choice of β, otherwise, has no role to play in the accuracy of estimation, it only ensures the possibility of finding the true maximum.

The choice of N again has no direct impact on the accuracy of estimation, a higher count of N only further ensures the probability that the dynamics has visited the unique maximum at least once. Since the knowledge of the initial condition is not available at this point of the chosen application, it cannot be calculatively determined how many iterations will be required to visit the maximum. However, if the shifting window size w is increased, N can be increased to avail sufficient number of shifts, since larger number of shifts will improve the probability of finding the S_{max}.

The window size w however needs to be chosen appropriately to ensure the accuracy of the estimation. Given the obvious notion that the accuracy would increase as the window size is increased, certain sizes of w has been observed (12 to 14 bits) beyond which the accuracy is stabilised to a steady outcome as can be seen in Fig. 5.5. Since the parameter is solved from the code of S_{max} appearing in the window, using a polynomial equation (5.3), addition of further bits to the window (increasing the size of w) will result in adding higher order polynomials that would contribute a nominal amount of information to the estimates, therefore, once the desired accuracy level is reached i.e. with w = [12,14], further addition of bits to the window may be ineffective as higher order polynomials may consume more time and processing power to converge to a solution.

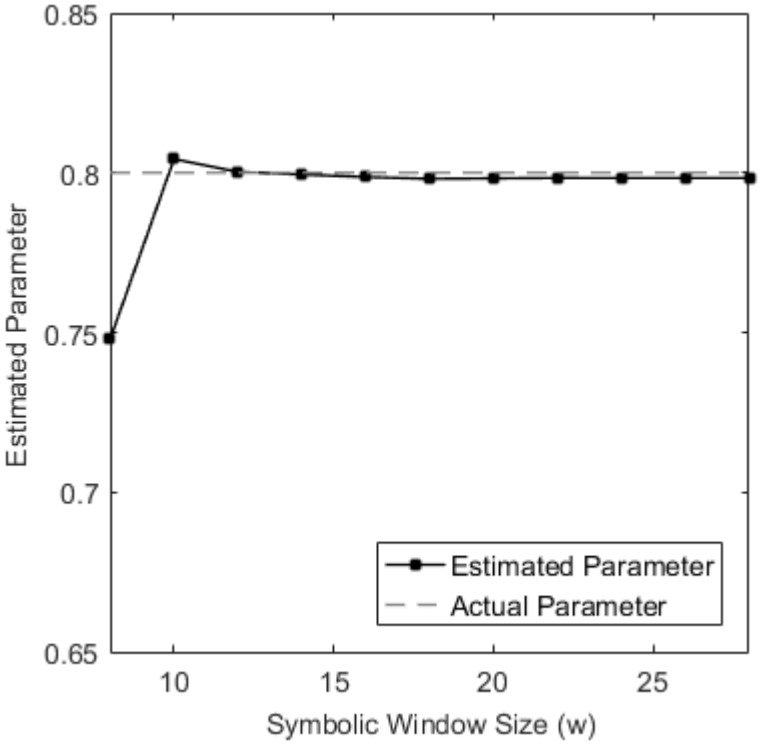


Fig. 5.5 Relationship between estimation accuracy and window size

Performance of such an estimation method may be affected by inherent noise in the physical circuitry, because in case of small overshoot the iterates may cross the map maximum and the corresponding S_{max} might get altered. In such situations the estimation algorithm can adapt to an averaging scheme operated on the frequently estimated outcomes. The averaging can improve the quality of the estimates by computing the mean value of the estimated parameters and updating the result.

5.2 Results: Parameter Estimation from Noisy Dynamics

The parameter estimation algorithm utilising noisy trajectories has been evaluated. To demonstrate working of the algorithm, an experimental condition has been chosen with an arbitrary initial condition: x_0 = 0.383 that was iterated through the TM with parameter μ = 0.715 for N = 50 iterations. The iterates were perturbed by dynamic addition of AWGN with SNR = 20 dB, and samples of η^m

trajectories were recorded for M = 200. For the chosen parameter value in this experiment the corresponding value of the non-zero fixed point is $x_f = 2\mu/(1 + 2\mu)$ = 0.588477, which may be compared with the estimated outcome of the algorithm.

According to the proposed technique, the Y^k solutions for the chosen case are the crossover points among the straight lines formed between n and $(n + 1)$ iterates. In Figs. 5.6 and 5.7, a collection of such Y^k solutions have been shown through histograms for the sampled iterates within two independent pair of time steps. Fig. 5.6 shows distribution of the Y^k solutions between n = 16 and 17. The mean value of the Y^k crossover points for this case is given as $\bar{Y}_{16}$ = 0.588973 and to realise the quality of the mean outcome, the standard deviation of the crossover solution points between n = 16 and 17 is found to be D_{16} = 0.090581.

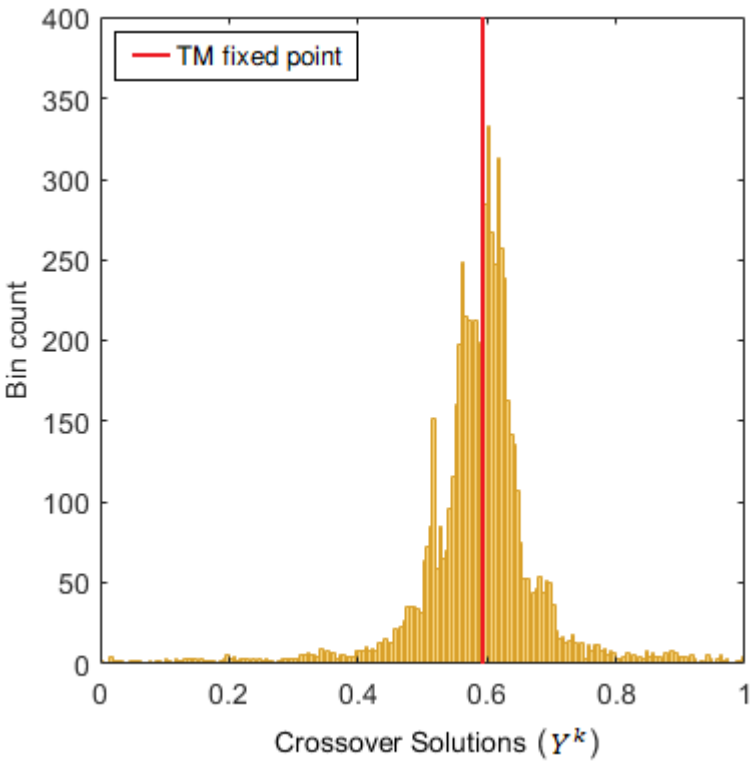


Fig. 5.6 Distribution of Y^ksolutions between n = 16 and 17

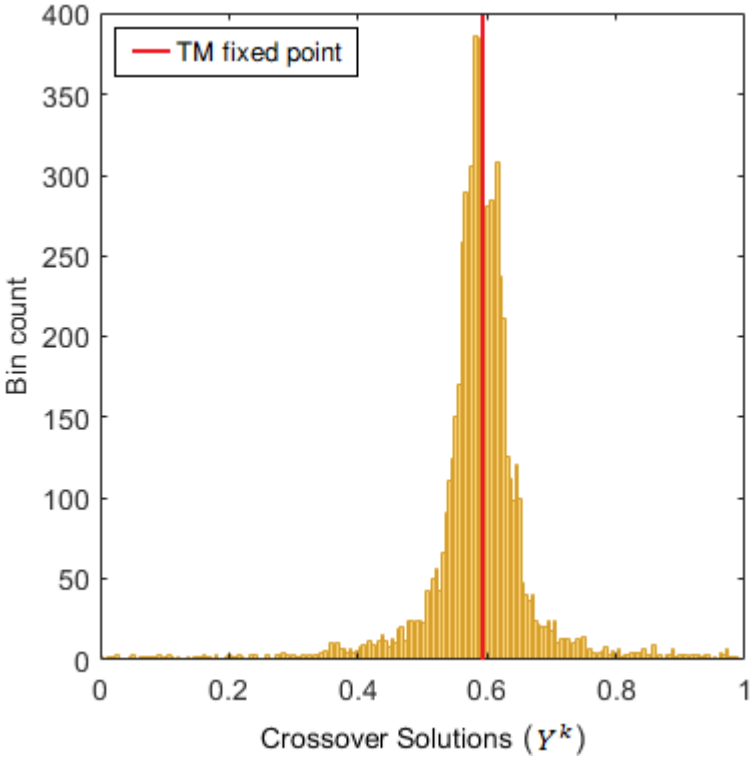


Fig. 5.7 Distribution of Y^k solutions between n = 20 and 21

In Fig. 5.7 the histogram distribution for Y^k solutions between time steps n = 20 and 21 has been shown with mean value of the solutions as $\bar{Y}_{20}$ = 0.588029 and the standard deviation of the distribution D_{20} = 0.079790. It can be noticed from the histograms, that the crossover solutions between the straight lines formed by the iterates of the two consecutive time steps are highly concentrated in the close neighbourhood of the actual fixed point x_f = 0.588477 of the map. Therefore, that the mean positions $\bar{Y}_n$ of the Y^k crossover points between every n and $(n+1)$ time steps are the closest estimates of the map fixed point. If a single estimate of all the $\bar{Y}_n$ estimates can be determined, the map fixed point can be ascertained more accurately. In Fig. 5.8, the $\bar{Y}_n$ values have been determined for all n timesteps and have been shown. It may be observed that, collectively the $\bar{Y}_n$ estimates are located closely around the actual fixed point x_f = 0.588477.

Further, mean of all the $\bar{Y}_n$ solutions were calculated as ξ = 0.588820 to determine the value of the map fixed point as precisely as possible.

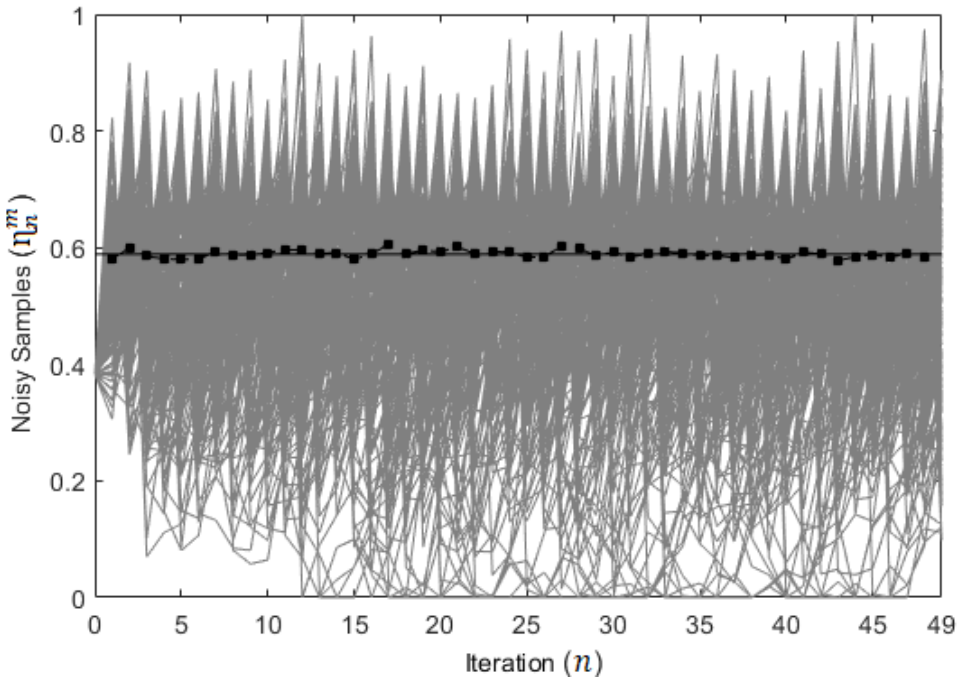


Fig. 5.8 The mean $\bar{Y}_n$ of crossover points (black-square legend)

As visualised through Fig. 5.8, the estimated fixed point ξ = 0.588820 is represented by the black straight line passing through the $\bar{Y}_n$ points. The quality of the mean value ξ from the collection of $\bar{Y}_n$ points was realised through standard deviation *SD* = 0.008620. The amount of error in the fixed-point estimation is given by 100(x_f - ξ) = -0.0343%, which is significantly low considering the effects of noise in the chaotic trajectories. From the value of ξ, as being the closest approximation of the TM non-zero fixed point x_f, the control parameter of the map has been estimated as μ' = 0.716027 that is as well the closest approximation of the actual parameter μ = 0.715 which was chosen for the experiment. The error in the parameter estimation is given as 100(μ - μ') = -0.1030%. It can be confirmed that under a harsh field of dynamical noise, the parameter value is closely estimated to the actual value with sufficient accuracy,

hence the method has been proved to be suitable for parameter estimation of the chaotic map from the noisy dynamics.

The estimation experiment was repeated for another arbitrary initial condition x_0 = 0.863281 and parameter μ = 0.90, and the trajectory was perturbed by dynamical noise of SNR = 30 dB. In this example the TM trajectory was iterated for N = 50 and each trajectory was sampled for M = 50. In Fig. 5.9 the noisy iterates and the estimated crossovers (fixed point) have been shown. The estimated fixed point from the mean of the crossover clusters $\bar{Y}_n$ was found to be ξ = 0.642892 whereas the actual fixed point for the chosen parameter value is x_f = 0.642857, the estimation proved to be in good agreement with the actual fixed point. Hence the estimated parameter was found to be μ' = 0.900130.

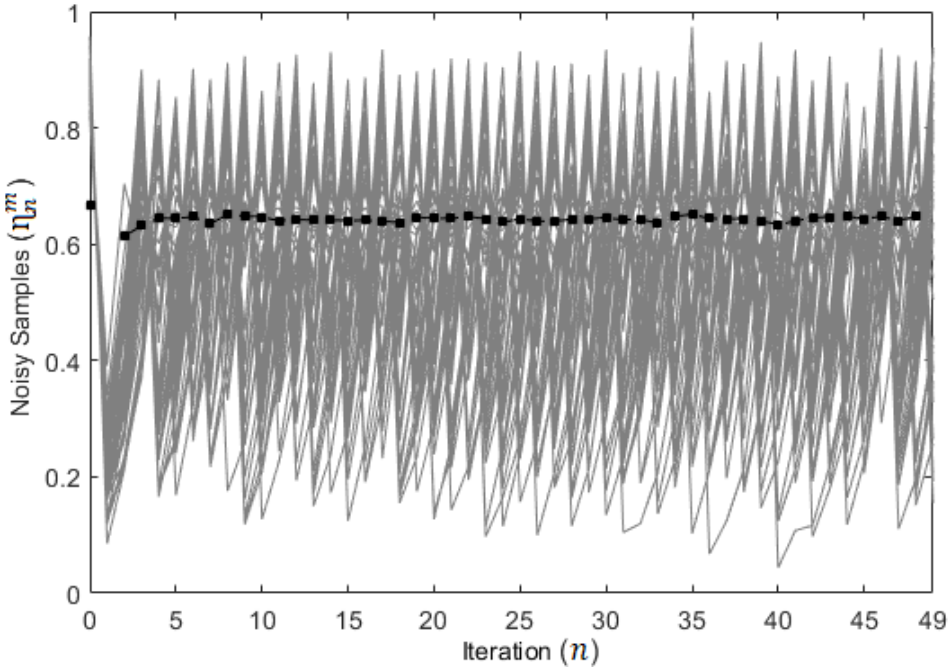


Fig. 5.9 Fixed point crossover estimates for SNR 30dB

To further realise the quality of the estimates using the proposed method parameter estimation using noisy dynamics, five independent cases of parameters μ = 0.95, μ = 0.90, μ = 0.85, μ = 0.80 and μ = 0.75 have been investigated. Due

to the statistical nature of the proposed algorithm, a confidence interval is determined for all estimation attempts representing the quality of the estimates. Each of the chosen cases of μ has been separately iterated through an arbitrary initial condition, for N = 50 iterations, and chaotic trajectory for each independent condition is repeatedly sampled M = 50 times. To determine the confidence level in the parameter estimations, the algorithm is operated repeatedly 25 times and μ' outcomes of each of the attempts have been recorded. The confidence interval depicted by the error bar and mean μ'_{mean} of Q = 25 independent estimation attempts $q = 1, 2, \dots, Q$, for each case of noise over a range of SNR values 10-30 dB have been determined.

To estimate the standard error-bar, the following calculation is applied for the mean of all the attempts for a given case of μ' outcome.

$$\mu'_{mean} = \frac{1}{Q}\sum_{q=1}^{Q} \mu'_q. \tag{5.4}$$

The upper and lower bounds of the 95% confidence interval is calculated respectively using:

$$\text{95\% Confidence Interval} = \mu'_{mean} \pm 1.96\left(\frac{\mu'_{SD}}{\sqrt{Q}}\right), \tag{5.5}$$

where, μ'_{SD} is the standard deviation of Q estimation attempts.

In Figs. 5.10 – 5.14, the quality of the estimated parameter has been shown through both μ'_{mean} and 95% confidence interval for the chosen cases of μ = 0.95, μ = 0.90, μ = 0.85, μ = 0.80 and μ = 0.75 respectively. The mean value of all the estimations is close to the actual μ belonging within the 95% confidence interval from SNR 15 dB onwards and the error bar reduces gradually.

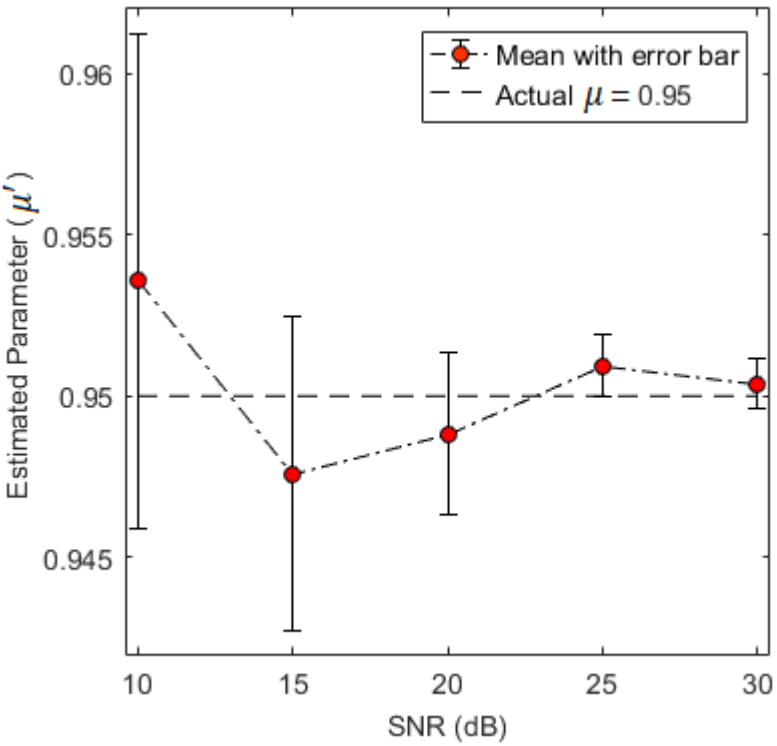


Fig. 5.10 Estimated parameter error bar plot for μ = 0.95

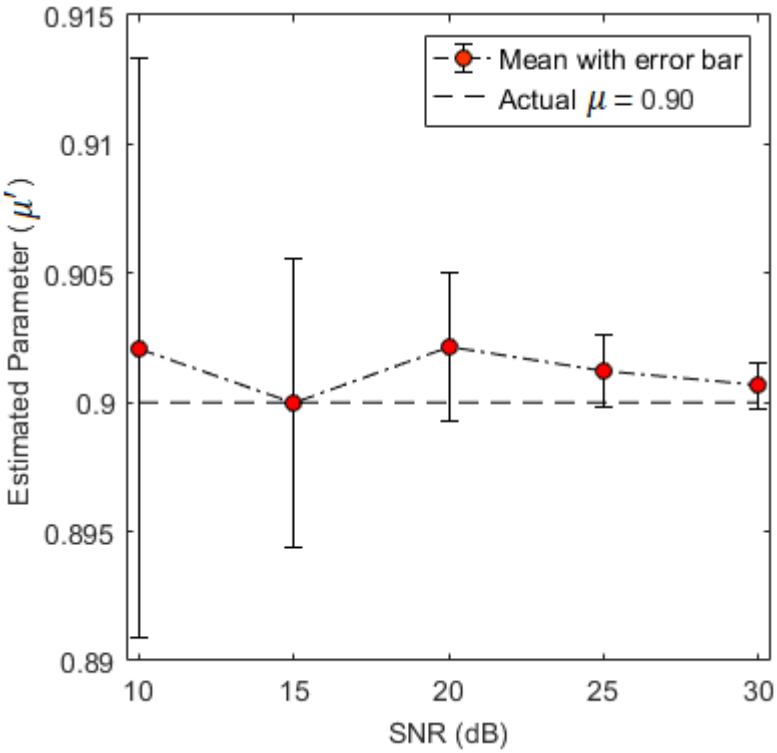


Fig. 5.11 Estimated parameter error bar plot for μ = 0.90

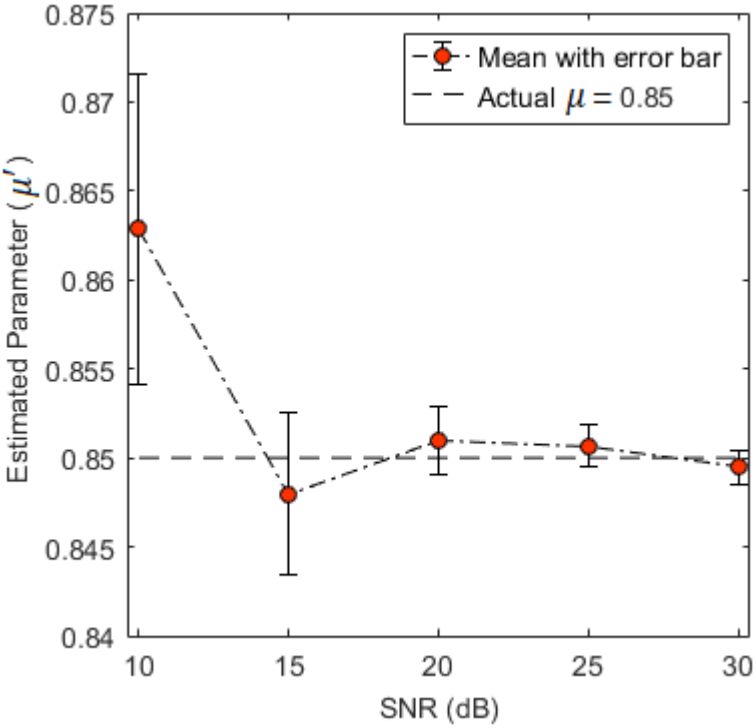


Fig. 5.12 Estimated parameter error bar plot for μ = 0.85

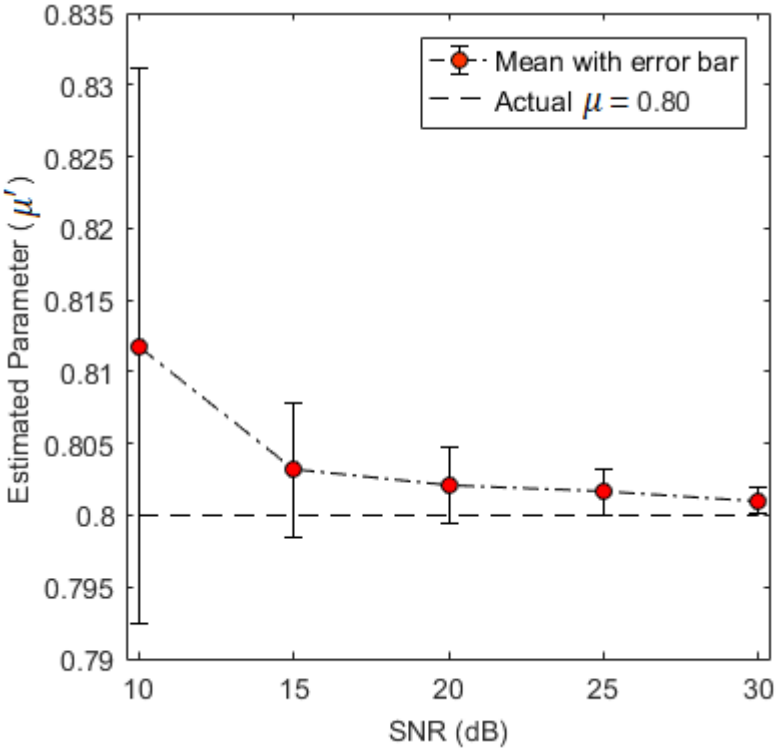


Fig. 5.13 Estimated parameter error bar plot for μ = 0.80

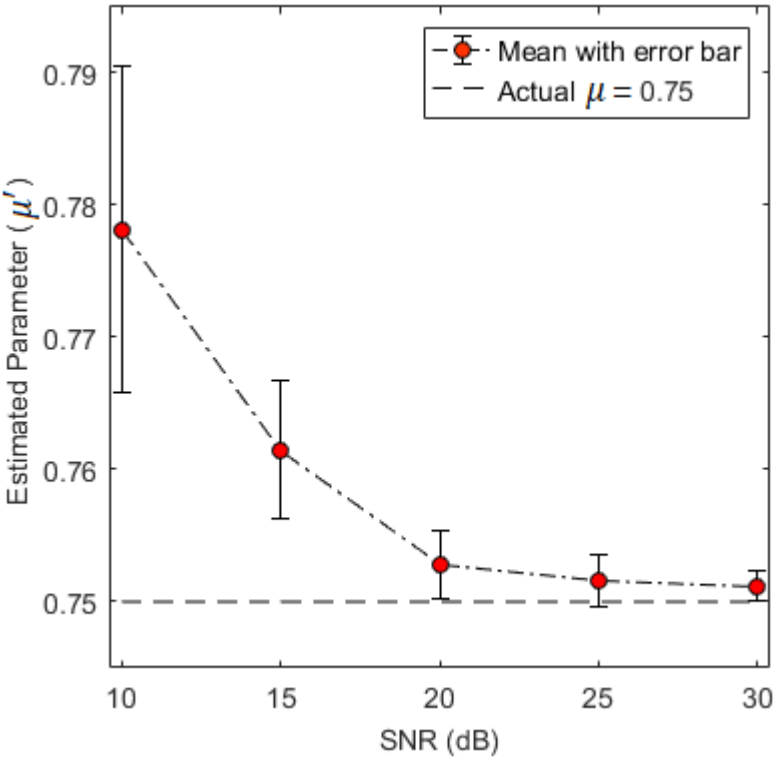


Fig. 5.14 Estimated parameter error bar plot for μ = 0.75

It can be noticed from the above error bar plots that the estimated outcomes are gradually deviated away from the actual μ for SNR values 10 dB or less. Hence, as a condition to utilise noise for parameter estimation, with slightly improved SNR values better results can be achieved as relatively lower noise may still preserve the qualitative properties of the TM dynamics. In harsher noisy conditions it has been previously investigated and established that the properties of the dynamical system are barely preserved [57].

The aim of the work is to establish the approaches for the correct identification of the non-ideal parameter so that it can be utilised in initial condition estimation. Due to the harsh characteristic of dynamical noise, the trajectories of the initial conditions might get severely affected. Therefore, the parameter estimation method has been further checked for the trajectories generated with a set of initial conditions perturbed by a certain degree of noise. In the following experiments

the set of 256 initial points have been iterated up to N = 32, with a parameter value μ = 0.715 and the trajectories have been perturbed by AWGN with SNR = 20 dB. The estimates were performed on M = 50 samples for each of the trajectories of 256 initial conditions. The map parameter was separately estimated from each of the perturbed trajectories.

In Figs. 5.15 – 5.17 the parameter values estimated from individual noisy trajectories generated by each initial condition within the state space have been shown for a range of SNR levels. Fig. 5.15 shows a perturbation by AWGN of SNR = 20 dB. Through the parameter estimation technique proposed here, the statistical trend of the estimated parameter values (roughly within range 0.71 – 0.73) are found to be close to the actual parameter value μ = 0.715 of the TM with which the trajectories have been generated.

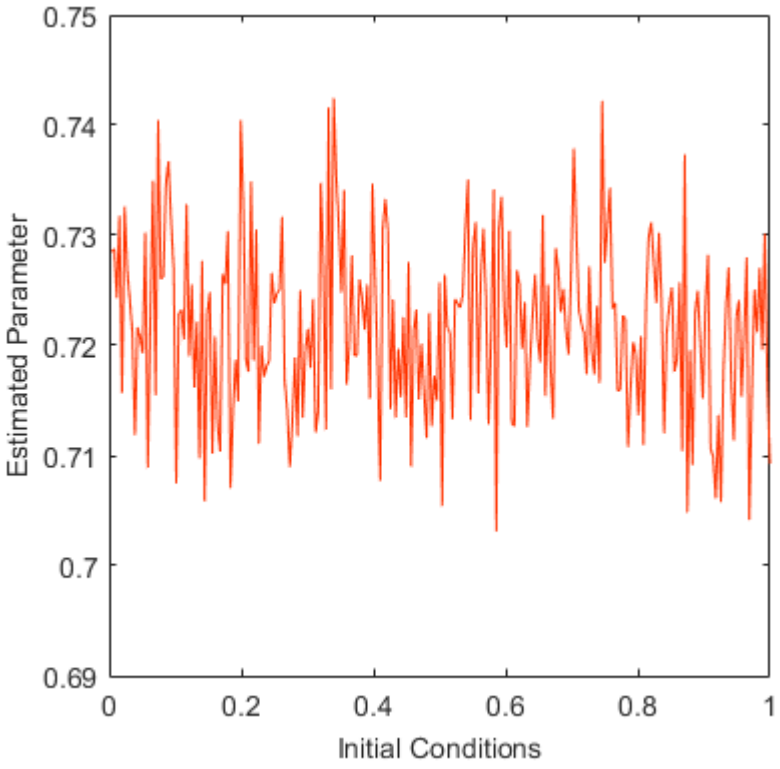


Fig. 5.15 Estimated parameter for all inputs (SNR = 20 dB)

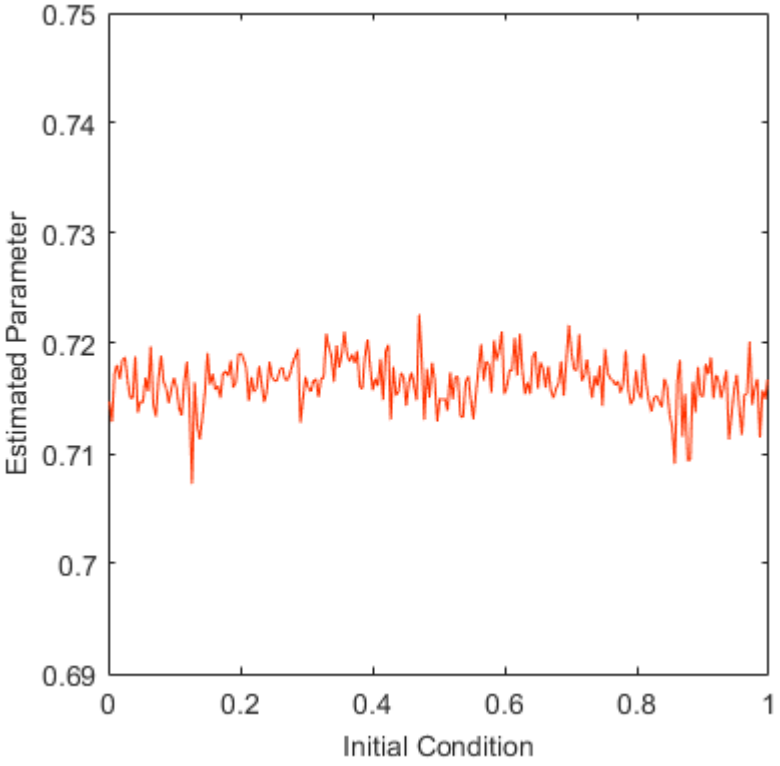


Fig. 5.16 Estimated parameter for all inputs (SNR = 25 dB)

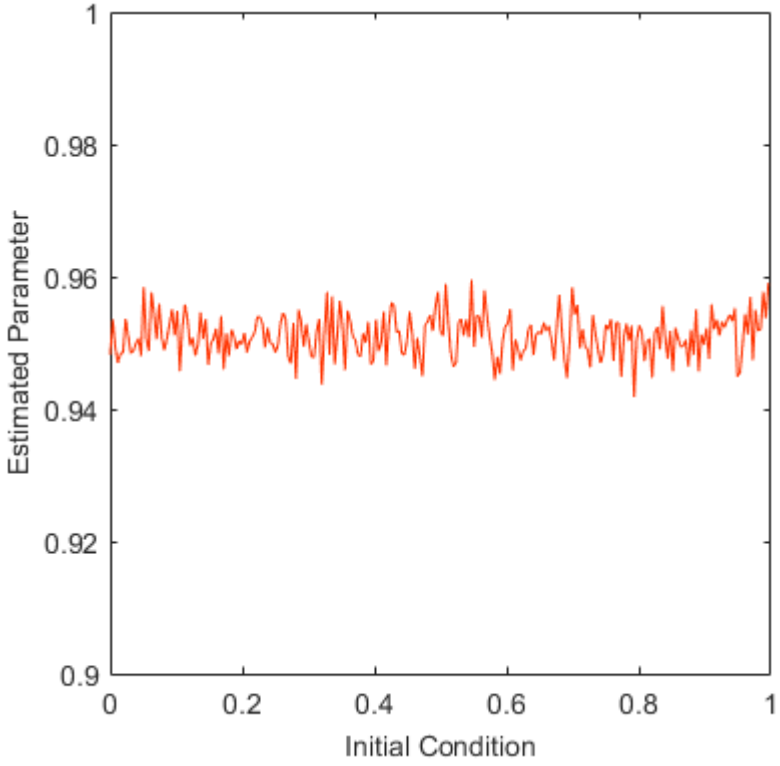


Fig. 5.17 Estimated parameter for all inputs (SNR = 30 dB)

The same experiment has been repeated with a slightly better SNR = 25 dB. The estimated parameter from the noisy trajectories has been shown in Fig. 5.16. The parameter estimates for most of the points are quite close to the actual parameter value μ = 0.715 as the estimates belong in the range 0.712 – 0.718. Another independent case with test parameter value μ = 0.95 and an SNR level of 30 dB has been considered for estimation. The estimated parameter from the noisy trajectories of the set of initial conditions has been shown in Fig. 5.17, the estimated parameters belong within range 0.945-0.955 (code for entire set of initial condition is provided in Appendix 2.13).

Even though noise levels with SNR = 20 dB and beyond is considered as moderate in general, such levels of noise may have drastic effects on the chaotic trajectories as the noise itself is dynamically multiplied through the chaotic function. Therefore, retrieving meaningful information such as map parameter becomes difficult. However, the proposed method of determining the map fixed points from the collection of crossovers has been proved to be useful for the approximation of the parameter value of TM.

In several cases of harsher conditions of dynamical noise (SNR = 10 dB or less), the system might depart from normal distribution [46], as the noise is propagated through dynamics. Due to the behaviour of the function corrupted by noise, some systematic error might get introduced, that may affect the statistical estimates. It is straightforward to deal with the random error using statistical methods compared to the systematic errors, as the source and behaviour of the systematic error might not always be known and might not exhibit normally distributed traits.

The proposed parameter estimation method from the noisy dynamics is found to be robust for the SNR 15 dB and beyond. In case of practical circuitry, the noise level may usually be expected to be better than 15 dB, in such conditions the estimations will be even better using the crossover-oriented algorithm. Despite the robustness of this estimation approach, currently the proposed algorithm utilises the real valued noisy iterates. However, in future, there could be further scopes to develop the method suitably for symbolic dynamics which will enhance the system resources further while maintaining the desired robustness.

6 CONCLUSION AND FUTURE SCOPES

Considering the chaotic dynamics approach for signal measurement, in this work, solutions to the parameter estimation of the implemented chaotic function have been presented. Tent map (TM) has been selected as the suitable chaotic function for signal measurement; because of the dense distribution of points that can be realised through TM dynamics, holding unique correspondence between chaotic trajectories and initial condition. However, when the map is implemented in the electronic hardware domain, due to offsets and tolerances of the components, the parameter of the map cannot be maintained at the ideal value and the map partitions shift from the ideal positions causing the dynamics to deviate from the actual path. Consequentially, such deviations result into loss of correspondence when initial condition is estimated from the symbolic sequences through conventional binary to decimal conversion techniques. It has been realised that the knowledge of the non-ideal parameter may be utilised to reinstate the correspondence and improve the accuracy of the initial condition estimation. Therefore, parameter estimation is essential and possible methods have been investigated.

The previously available techniques used extensively long dynamical trajectories to estimate the parameter. Such approaches were mainly dedicated to the field of communication, where, acquiring millions of iterations from the computationally implemented maps were not a problem. However, for the signal measurement using electronic hardware, in this work, the knowledge of the dynamical properties in non-ideal conditions has been suitably utilised to formulate the parameter estimation techniques using significantly less number of iterations.

Two innovative techniques for parameter estimation have been proposed. One of the proposed approaches is the Kneading sequence search algorithm, which was achieved by operating a symbolic shifting window over the entire symbolic sequence generated with an initial condition for any non-ideal parameter. The symbolic sequence corresponding to the map maximum was determined by comparing the GONs of each shift of the window. The maximum sequence was then converted to the Kneading sequence and finally the parameter was estimated by solving the difference equation that was established with the map critical point. The estimated parameter values were found to be considerably accurate with estimation error approximately under 0.5%. Also, the estimation was achieved with reduced number of iterations (200) compared to the conventional techniques.

The presence of noise in the chaotic systems result in highly digressing trajectories leading to difficulties in determining the desired information of the actual trajectory. Numerous researches have confirmed the importance of the knowledge of map parameter, such that the actual dynamical trajectory can be discerned from the noisy ones. This prompted the investigation of another approach that utilises the distribution of the noisy chaotic trajectories to estimate the map parameter. From the properties of the noisy dynamics of TM, as has been simulated in this work under various noise levels, unique crossovers between the trajectories have been observed at the close neighbourhood of the non-zero fixed point of the TM. The presence of noise in the system has in fact, enhanced the probability of finding the crossovers within the perturbed trajectories, since, noise causes the dynamics to be highly distributed over the state space. The proposed parameter estimation technique utilised a set of (50) sampled

trajectories of the same dynamics iterated up to 50 times. The crossovers appeared between the iterates of each consecutive pair of time steps were determined by solving linear equations, as the iterates were represented through Cartesian coordinate system, where the *x*-axis depicted the time step and the *y*-axis depicted the magnitude of the iterate. The concentration of the crossovers between the sampled trajectories was statistically located by taking average of the solutions. Since the location of crossovers corresponds to the non-zero fixed point of the TM, the map parameter has been estimated from the knowledge of the non-zero fixed point. For such a statistical approach applied over dynamically affected noisy time series, the actual parameter was found to be contained within 95% confidence interval of the estimation for the SNR 15 dB onwards, and with the standard deviation of the estimates was found to be between 0.07 to 0.09.

Both the proposed approaches can be easily implemented through programs in the computation domain and in electronic hardware such as field programmable gate array (FPGA) and microcontrollers. For the desired application of signal measurement, the techniques can be coupled directly with the initial condition estimation algorithms to accurately determine the starting point or the input signal from the dynamics. This development in the parameter estimation methods in accompaniment with initial condition methods can be considered as a step forward in the development of a chaotic ADC, and the entire measurement system may be implemented in a single chip package.

6.1 Future scopes

The knowledge of the chaotic dynamics and the methods devised for parameter estimation in this work can be broadly applied in various related and independent

areas. Apart from the desired scope of signal measurement, the proposed parameter estimation approaches can also be utilised in other applications where chaotic maps are widely used e.g. for cryptography and encryption in communication technologies.

6.1.1 Chaotic Measurement System Implementation

To implement a TM-based chaotic measurement system as a standalone technology, either of the two proposed parameter estimation techniques can be employed. If the amount of noise in the implemented system hardware ranges from SNR values lower than 30 dB, then the crossover-oriented method can be applied, as the method efficiently determines parameter from the noisy dynamics with SNR as low as 15 dB. On the other hand, if the noise in the circuit is significantly low, the shifting window technique can be utilised as the method can be directly operated in the symbolic domain. The accuracy of the estimates affected by a small amount of noise can be further optimised by employing moving average algorithm which will continue to compute the average of the estimated outcomes from time to time.

Successful implementation of a chaotic measurement system as an ADC is expected to save a considerable amount of resources and reduce the design complexity significantly because a single block of chaotic map can be used as a quantisation unit. Following (see Fig. 6.1) is the functional block diagram of the chaotic measurement system. The analogue implementation of the TM can be adapted from the schematic circuit which was first proposed by Campos-Cantón et al [14] and later utilised by Sanjin Berbercick [19] and Basu et al [24] to study the application further. The analogue TM comprise of simple electronic circuit

involving op-amps with positive and negative gains for the mathematical operations of two branches of the equation.

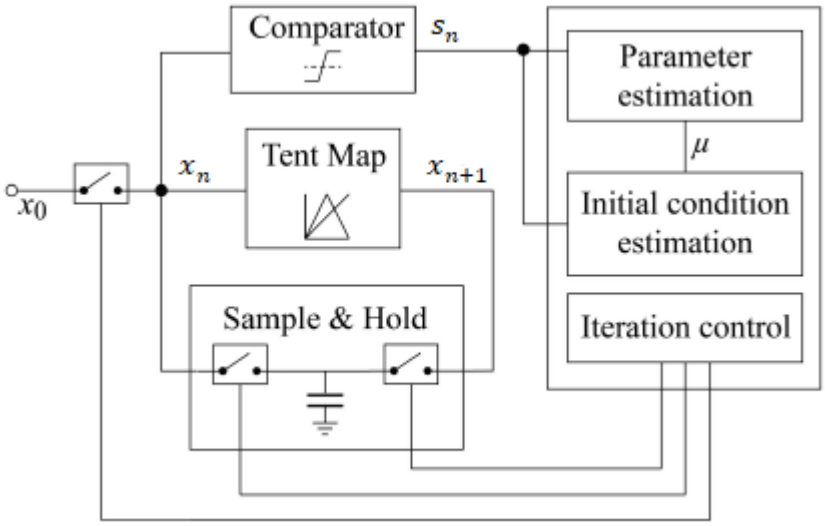


Fig. 6.1 Functional block diagram of the measurement system [24]

The feedback process of the iterative dynamics can be performed by incorporating sample-and-hold circuits in both input and output stages controlled by suitable clocking mechanism generated by a digital controller (microcontroller or FPGAs). The clocking of the input-output sample-and-hold stages for the iteration cycles has to be at par with the sampling of the iterates on each time step, and therefore can be generated from the same digital controller where the estimation algorithms will be performed so that the iterates can be sampled while performing the computations in parallel. The symbolic output for each iteration can be generated by including a comparator referenced to the threshold of 0.5V at the input stage of the feedback loop. Based on the estimated parameter, the initial condition can be estimated correctly, leading to successful recovery of the input value.

Also, there could be additional requirements regarding the hardware implementation of the chaotic system that would demand further investigation before the technology can be released off the shelf. Following factors have been identified that need addressing, e.g. symbolic approach for noisy dynamics, shift of the map critical point.

6.1.1.1 Symbolic Approach for Noisy Chaos

Keeping in mind the noisy conditions that may arise in the hardware implemented chaotic maps; the parameter estimation approach utilising the noisy trajectories needs to be formulated for the symbolic dynamics as well. Since the proposed parameter estimation approach is based on real iterates, it might use an additional ADC to gather real valued noisy iterates into the processing domain. Modifying the approach for symbolic dynamics will offer a robust solution as additional ADCs will no longer be needed and noise can still be utilised to estimate the parameter directly from the symbolic dynamics.

6.1.1.2 Shift of Critical Point

The TM is a piecewise linear function and the two piecewise stretching and folding restrictions of the map are defined about the critical point or midpoint of the map. When the map is implemented in physical hardware, the midpoint is also prone to shift from the ideal value of 0.5 which might cause the dynamics to diverge from the desired trajectory.

A broad study must be conducted to understand the effect of the midpoint shift on the dynamics and suitable solutions to address the problem should be investigated. However, alternatively the problem can also be addressed by utilising a slightly modified version of TM called skew tent map [58]. The

primary advantage of the skew tent maps is that, the map is completely defined by a single parameter that is the critical point of the map. Unlike tent maps, the skew tent maps are always of full height as there is no reduction of height due to the non-ideal parameter. Any change in the parameter will only result into shift in the primary partition or the map critical point causing the map to appear as asymmetric or skewed. Since the dynamics is controlled by a single parameter, it is beneficial to utilise skew tent maps as a signal quantiser of chaotic ADCs, and accordingly the parameter estimation and initial condition estimation methods can be modified.

6.1.2 Applications of Chaos in Encryption

The most common area of application of chaotic dynamics is encryption. Since chaotic trajectories apparently appear to be random, information can be protected by encrypting through the dynamics of a chaotic map. The map parameter is often utilised as a cipher key that must be used during the process of decryption of the actual information from the available chaotic dynamics. Therefore, parameter estimation of the map from a chaotic trajectory representing the encrypted information is one of the essential steps. TM is a widely used candidate for encryption as the map generates dense chaotic trajectories for a wide range of parameter. Due to such a robust chaotic distribution of the TM, information can be chaotically mutated into completely different and random data such that hacking of the information can be prevented. The original data is decrypted from the chaotic dynamics using the knowledge of the parameter. Therefore, the proposed parameter estimation method can be a useful addition to the decryption process of the chaotically encrypted data. There are other approaches for encryption that use single chaotic map or coupled maps as the

encryption function, accordingly, the proposed parameter estimation methods can be modified complying with the dynamical setting of the type of chaotic function chosen.

REFERENCES

[1] Bentley, J. P. (2005). *Principles of measurement systems* (4th ed.). Pearson Prentice Hall

[2] Sheingold, D. H., Analog Devices. (1986). *Analog-digital conversion handbook (3rd ed.).* London; Englewood Cliffs:; Prentice-Hall.

[3] Mather, P. J. (n.d.). 6. Mixed-Signal Circuit Structures. Lecture presented at *NIE2203 Electronics 2 Lecture* in University of Huddersfield, Huddersfield

[4] Silva, J., Moon, U., Steensgaard, J., & Temes, G. C. (2001). Wideband low-distortion delta-sigma ADC topology. *Electronics Letters, 37*(12), 737. 10.1049/el:20010542

[5] Hong, X., Yang, C., & Zhang, X. (2017). An energy-efficient SAR ADC with a partial-monotonic capacitor switching technique. *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing,* 2050-2054. 10.1109/IAEAC.2017.8054377

[6] Chiu, Y., Gray, P. R., & Nikolic, B. (2004). A 14-b 12-MS/s CMOS pipeline ADC with over 100-dB SFDR. *IEEE Journal of Solid-State Circuits, 39*(12), 2139-2151. 10.1109/JSSC.2004.836232

[7] Kimura, H., Matsuzawa, A., Nakamura, T., & Sawada, S. (1993). A 10-b 300-MHz interpolated-parallel A/D converter. *IEEE Journal of Solid-State Circuits, 28*(4), 438-446. 10.1109/4.210026

[8] Walden, R. H. (1999). Analog-to-digital converter survey and analysis. *IEEE Journal on Selected Areas in Communications, 17*(4), 539-550. 10.1109/49.761034

[9] Bashir, S., Ali, S., Ahmed, S., & Kakkar, V. (2016). Analog-to-digital converters: A comparative study and performance analysis. *2016 International Conference on Computing, Communication and Automation (ICCCA), Noida*, 999-1001. 10.1109/CCAA.2016.7813861

[10] Kennedy, M. P. (1995). a nonlinear dynamics interpretation of algorithmic a/d conversion. *International Journal of Bifurcation and Chaos, 5*(3), 891-893. 10.1142/S0218127495000685

[11] Berliner, L. M. (1992). Statistics, probability and chaos. *Statistical Science, 7*(1), 69-90. 10.1214/ss/1177011444

[12] Cvitanovic, P., Artuso, R., Mainieri, R., Tanner, G. and Vattay, G. *Chaos: Classical and Quantum* (Niels Bohr Institute, Copenhagen 2016) [online: Chaosbook.org].

[13] Gilmore, R., and Lefranc, M. (2002) "Discrete Dynamical Systems: Maps," *The Topology of Chaos, 1st ed.* New York, NY, USA: JW&Sons, 40-53

[14] Campos-Cantón, I., Campos-Cantón, E., Murguía, J. S., & Rosu, H. C. (2009). A simple electronic circuit realization of the tent map. *Chaos, Solitons and Fractals, 42*(1), 12-16. 10.1016/j.chaos.2008.10.037

[15] Suneel, M. (2006). Electronic circuit realization of the logistic map. *Sadhana, 31*(1), 69-78. 10.1007/BF02703801

[16] Ott, E. (1981). Strange attractors and chaotic motions of dynamical systems. *Reviews of Modern Physics, 53*(4), 655-671. 10.1103/RevModPhys.53.655

[17] Kapitaniak, T., Zyczkowski, K., Feudel, U., & Grebogi, C. (2000). Analog to digital conversion in physical measurements. *Chaos, Solitons and Fractals, 11*(8), 1247-1251. 10.1016/S0960-0779(99)00003-X

[18] Litovski, V., Andrejevic, M., & Nikolic, M. (2006). Chaos based analog-to-digital conversion of small signals. *2006 8th Seminar on Neural Network Applications in Electrical Engineering, Belgrade, Serbia & Montenegro,* 173-176. 10.1109/NEUREL.2006.341205

[19] Berberkic, S. (2014) Measurement of small signal variations using one-dimensional chaotic maps. *Doctoral thesis*, University of Huddersfield

[20] Radwan, A. G., Abd-El-Hafiz, S. K., & AbdElHaleem, S. H. (2014). An image encryption system based on generalized discrete maps. *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS), Marseille*, 283-286. 10.1109/ICECS.2014.7049977

[21] Arroyo, D., Alvarez, G., Li, S., Li, C., & Fernandez, V., (2009). Cryptanalysis of a new chaotic cryptosystem based on ergodicity. *International Journal of Modern Physics, B, 23*(5), 651–659.

[22] Metropolis, N., Stein, P. R., & Stein, M. L. (1973). On finite limit sets for transformations on the unit interval. *Journal of Combinatorial Theory, Series A, 15*(1), 25-44. 10.1016/0097-3165(73)90033-2

[23] Álvarez, G., Romera, M., Pastor, G., & Montoya, F. (1998). Gray codes and 1D quadratic maps. *Electronics Letters, 34*(13), 1304. 10.1049/el:19980950

[24] Basu, R., Dutta, D., Banerjee, S., Holmes, V., & Mather, P. (2017). An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map. *IEEE Transactions on Circuits and Systems I: Regular Papers* , *65*(7), 2221-2231. 10.1109/TCSI.2017.2773202

[25] Banerjee, S., Yorke, J. A., & Grebogi, C. (1998). Robust chaos. *Physical Review Letters, 80*(14), 3049-3052. 10.1103/PhysRevLett.80.3049

[26] Arroyo, D., & Alvarez, G. (2014). Application of gray codes to the study of the theory of symbolic dynamics of unimodal maps. *Communications in Nonlinear Science and Numerical Simulation, 19*(7), 2345. 10.1016/j.cnsns.2013.11.005

[27] Xi, C., Yong, G. and Yuan, Y. (2009). A Novel Method for the Initial-Condition Estimation of a Tent Map. *Chinese Physics Letters, 26*(7), pp. 078202 - 1–3. 10.1088/0256-307X/26/7/078202

[28] Cong, L., Xiaofu, W., & Songgeng, S. (1999). A general efficient method for chaotic signal estimation. *IEEE Transactions on Signal Processing, 47*(5), 1424-1428. 10.1109/78.757236

[29] Dutta, D., Basu, R., Banerjee, S., Holmes, V., & Mather, P. (2018). Parameter estimation for 1D PWL chaotic maps using noisy dynamics. *Nonlinear Dynamics*, *94*(4), 2979-2993. 10.1007/s11071-018-4538-x

[30] Taylor, J. R., & Teĭlor, D. (1997). *An introduction to error analysis: The study of uncertainties in physical measurements* (2nd ed.). Sausalito, Calif: University Science Books.

[31] Texas Instruments (1994). *Principles of Data Acquisition and Conversion* (Application Report No. SBAA051A). Retrieved from Texas Instruments website: http://www.ti.com/lit/an/sbaa051a/sbaa051a.pdf

[32] Biswas, S. S., Bindra, M., Jain, V., & Gokhale, P. (2015). Evaluation of imprecision, bias and total error of clinical chemistry analysers. *Indian Journal of Clinical Biochemistry, 30*(1), 104-108. 10.1007/s12291-014-0448-y

[33] Nauta, B., & Venes, A. G. W. (1995). A 70-MS/s 110-mW 8-b CMOS folding and interpolating A/D converter. *IEEE Journal of Solid-State Circuits, 30*(12), 1302-1308. 10.1109/4.482155

[34] Lorenz, E. (n.d.). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences, 20*, 130-141

[35] Strogatz, S. H. (2000). *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry and engineering*. Cambridge, Mass: Westview

[36] Bryant, P., & Brown, R. (1990). Lyapunov exponents from observed time series. *Physical Review Letters, 65*(13), 1523-1526. 10.1103/PhysRevLett.65.1523.

[37] Pastor, G., Romera, M., & Montoya, F. (1997). A revision of the lyapunov exponent in 1D quadratic maps. *Physica D: Nonlinear Phenomena, 107*(1), 17-22. 10.1016/S0167-2789(97)00057-2

[38] Bacaër, N. (2011). *A short history of mathematical population dynamics*. Springer Science & Business Media.

[39] Arroyo, D., Amigó, J. M., Li, S. J., & Alvarez, G. (2010). On the inadequacy of unimodal maps for cryptographic applications. *11th Spanish Meeting on Cryptology and Information Security (RECSI 2010),* 2010. Tarragona, Spain

[40] Ilyas, A., Luca, A., & Vlad, A. (2012). A study on binary sequences generated by tent map having cryptographic view. *2012 9th International Conference on Communications (COMM)*, 2012.

[41] Collet, P., & Eckmann, J. (2009). Modern birkhäuser classics*: Iterated maps on the interval as dynamical systems.* Birkhäuser Boston

[42] Bollt, E., Standford,T., Lai, Y., and Życzkowski, K. (2001). What symbolic dynamics do we get with a misplaced partition? on the validity of threshold crossings analysis of chaotic time-series. *Physica D: Nonlinear Phenomena, 154*(3-4), 259-286. 10.1016/S0167-2789(01)00242-1

[43] Wu, X., Hu, H., & Zhang, B. (2004). Parameter estimation only from the symbolic sequences generated by chaos system. *Chaos, Solitons and Fractals, 22*(2), 359-366. 10.1016/j.chaos.2004.02.008

[44] Amigó, J. M., Elizalde, S., & Kennel, M. B. (2008). Forbidden patterns and shift systems. *Journal of Combinatorial Theory, Series A, 115*(3), 485-504. 10.1016/j.jcta.2007.07.004

[45] Arroyo, D., Alvarez, G., & Amigó, J. M. (2009). Estimation of the control parameter from symbolic sequences: Unimodal maps with variable critical point. Chaos: *An Interdisciplinary Journal of Nonlinear Science, 19*(2), 023125-023125-9. 10.1063/1.3155072

[46] Strumik, M., & Macek, W. M. (2008). Influence of dynamical noise on time series generated by nonlinear maps. *Physica D: Nonlinear Phenomena, 237*(5), 613-618. 10.1016/j.physd.2007.10.002

[47] Orzeszko, W. (2008). The new method of measuring the effects of noise reduction in chaotic data. *Chaos, Solitons and Fractals, 38*(5), 1355-1368. 10.1016/j.chaos.2007.06.059

[48] Kantz, H., & Schreiber, T. (2003). *Nonlinear time series analysis* (2nd ed.). GB: Cambridge University Press

[49] Kostelich, E. J., & Schreiber, T. (1993). Noise reduction in chaotic time-series data: A survey of common methods. *Physical Review E, 48*(3), 1752-1763. 10.1103/PhysRevE.48.1752

[50] Kostelich, E. J., & Yorke, J. A. (1990). Noise reduction: Finding the simplest dynamical system consistent with the data. *Physica D: Nonlinear Phenomena, 41*(2), 183-196. 10.1016/0167-2789(90)90121-5

[51] Casdagli, M. (1989). Nonlinear prediction of chaotic time series. *Physica D: Nonlinear Phenomena, 35*(3), 335-356. 10.1016/0167-2789(89)90074-2

[52] Abarbanel, H. D. I., & SpringerLink (Online service). (1996). *Analysis of observed chaotic data (1st ed.).* New York: Springer. 10.1007/978-1-4612-0763-4

[53] Badii, R., Broggi, G., Derighetti, B., Ravani, M., Ciliberto, S., Politi, A., & Rubio, M. A. (1988). Dimension increase in filtered chaotic signals. *Physical Review Letters, 60*(11), 979-982. 10.1103/PhysRevLett.60.979

[54] Schreiber, T. (1993). Determination of the noise level of chaotic time series. *Physical Review E, 48*(1), R13-R16. 10.1103/PhysRevE.48.R13

[55] Davies, M. (1994). Noise reduction schemes for chaotic time series. *Physica D: Nonlinear Phenomena, 79*(2), 174-192. 10.1016/S0167-2789(05)80005-3

[56] Grassberger, P., Hegger, R., Kantz, H., Schaffrath, C., & Schreiber, T. (1993). On noise reduction methods for chaotic data. Chaos: *An Interdisciplinary Journal of Nonlinear Science, 3*(2), 127-141. 10.1063/1.165979

[57] Ott, E., Yorke, E. D., & Yorke, J. A. (1985). A scaling law: How an attractor's volume depends on noise level. *Physica D: Nonlinear Phenomena, 16*(1), 62-78. 10.1016/0167-2789(85)90085-5

[58] Wang, K., Pei, W., Hou, X., Shen, Y., & He, Z. (2009). Symbolic dynamics approach to parameter estimation without initial value. *Physics Letters A, 374*(1), 44-49. 10.1016/j.physleta.2009.10.021

APPENDIX 1: PUBLICATIONS

List of peer-reviewed articles

1.1 Parameter estimation for 1D PWL chaotic maps using noisy dynamics

1.2 An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map

Appendix 1.1

This article has been published in the Nonlinear Dynamics and was first online on 18 September 2018 and can be found online at: 10.1007/s11071-018-4538-x

Parameter Estimation for 1D PWL Chaotic Maps Using Noisy Dynamics

D. Dutta, R Basu, S. Banerjee, V. Holmes and P. Mather

D. Dutta, R. Basu, V. Holmes and P. Mather are with the Engineering and Technology Department, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield, W. Yorks., UK, HD1 3DH (email: dhruba.dutta@hud.ac.uk, rajlaxmi.basu@hud.ac.uk, v.holmes@hud.ac.uk, p.j.mather@hud.ac.uk).

S. Banerjee is with the Department of Physical Sciences, Indian Institute of Science Education & Research, Kolkata, Mohanpur Campus, Nadia-741246, India (email: soumitro@iiserkol.ac.in).

Reference
Dutta, D., Basu, R., Banerjee, S., Holmes, V., & Mather, P. (2018). Parameter estimation for 1D PWL chaotic maps using noisy dynamics. *Nonlinear Dynamics*, *94*(4), 2979-2993.

Appendix 1.2

This article has been published in the Transactions in Circuits and Systems – I on 7 December 2017 and can be found online at: 10.1109/TCSI.2017.2773202

An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map

R Basu, D. Dutta, S. Banerjee, V. Holmes and P. Mather

R. Basu, D. Dutta, V. Holmes and P. Mather are with the Engineering and Technology Department, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield, W. Yorks., UK, HD1 3DH (email: rajlaxmi.basu@hud.ac.uk, dhruba.dutta@hud.ac.uk, v.holmes@hud.ac.uk, p.j.mather@hud.ac.uk).

S. Banerjee is with the Department of Physical Sciences, Indian Institute of Science Education & Research, Kolkata, Mohanpur Campus, Nadia-741246, India (email: soumitro@iiserkol.ac.in).

Reference
Basu, R., Dutta, D., Banerjee, S., Holmes, V., & Mather, P. (2018). An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *65*(7), 2221-2231.

APPENDIX 2: MATLAB CODES

MATLAB codes for simulating the map behaviours and verifying proposed parameter estimation algorithms are listed.

2.1 Logistic map

2.2 LM bifurcation diagram

2.3 Bitshift map

2.4 BM bifurcation diagram

2.5 Tent map

2.6 TM bifurcation diagram

2.7 TM cobweb

2.8 Gray Ordering Number (GON)

2.9 Shifting window

2.10 Kneading sequence search algorithm for single input

2.11 Kneading sequence search algorithm for entire input dataset

2.12 Parameter estimation from noisy dynamics for single input

2.13 Parameter estimation from noisy dynamics for entire input dataset

Appendix 2.1: Logistic Map (LM)

Program for Logistic Map (LM)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Logistic Map generated for  ----------------------%%
%%-------------------- a set of initial conditions ----------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format longe

clear;                   % clears variables
clf;

Map_partition = 0.50;    % primary partition of the map
iteration = 10;           % setting number of iterations

Resolution = 8;          % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                          % input range [0,1] lower limit = 0
x_max = 1;                      % input range [0,1] upper limit = 1
xNew = x;                       % starting initial condition

N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions
                                  % within [0,1]

Real_Trajectories = zeros(N,iteration); % stores all iterates for N initial
                                        % conditions
Symbolic_Trajectories = zeros(N,iteration); % stores symbolic sequences for
                                              % N initialconditions
```

```
parameter = 0.9;                % sets the peak height of the map

for i = 1:N                     % runs for N number of initial conditions

    x1 = x;                     % copy initial condition to input variable

    for n = 1:iteration         % runs map for all iterations
        x2 = 4*parameter*x1*(1-x1); % map operation, determine next iterate
        if x1 <= Map_partition      % condition for when x1 < 0.5
            op = 0;                 % store symbol as 0
        elseif x1 > Map_partition   % condition for when x1 >= 0.5
            op = 1;                 % store symbol as 1
        end
        Real_Trajectories(i,n) = x1;      % stores the real ietrate
        Symbolic_Trajectories(i,n) = op;  % stores the symbol
        x1 = x2;                           % replaces old x1 with new x2

    end

    x = (x + increment);            % increases x by one step for next
                                      % initial condition
end

%---------- Map Plot -------------------------------------------------
plot(Real_Trajectories(:,1),Real_Trajectories(:,2)); % plot map
set(gca,'xlim',[0 1]);                               % set axis views
set(gca,'ylim',[0 1]);
axis square;
```

Appendix 2.2: LM Bifurcation Diagram

Program for LM bifurcation diagram

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Logistic Map bifurcation    ----------------------%%
%%-------------------- diagram                     ----------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Npre = 250;         % no. of initial few iterates thrown away for clear view
Nplot = 100;                % no. of points in an iteration to be plotted
x = zeros(Nplot,1);         % x trajectory array initialised
parameter = zeros(Nplot,1);   % parameter array initialised


for r = 0.0:0.00025:1.0       % for parameters sweeping [0,1] range
     x(1) = 0.5;              % set initial condition as 0.5
    for n = 1:Npre            % iterate the map for up to Npre
        x(1) = 4*r*x(1)*(1-x(1)); % determine next iterate
    end,
    for n = 1:Nplot-1               % iterate the map for up to Npre
        x(n+1) = 4*r*x(n)*(1-x(n)); % map operation
    end
    plot(r*ones(Nplot,1), x, 'k.', 'markersize', 3); % ploting the iterates
    hold on;
end,

xlabel('μ');  ylabel('x_n');   % setting axis labels and viewing dimentions
set(gca, 'xlim', [0.5 1]);
set(gca, 'ylim', [0 1]);
```

Appendix 2.3: Bitshift Map (BM)

Program for Bitshift Map (BM)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Bitshift Map generated for  ----------------------%%
%%-------------------- a set of initial conditions ----------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format longe

clear;                  % clears variables
clf;

Map_partition = 0.50;   % primary partition of the map
iteration = 10;          % setting number of iterations

Resolution = 8;         % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                          % input range [0,1] lower limit = 0
x_max = 1;                      % input range [0,1] upper limit = 1
xNew = x;                       % starting initial condition

N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions
                                  % within [0,1]

Real_Trajectories = zeros(N,iteration); % stores all iterates for N initial
                                        % conditions
Symbolic_Trajectories = zeros(N,iteration); % stores symbolic sequences for
                                              % N initialconditions
```

```
parameter = 1;                % sets the peak height of the map

for i = 1:N                   % runs for N number of initial conditions

    x1 = x;                   % copy initial condition to input variable

    for n = 1:iteration       % runs map for all iterations

        if x1 <= Map_partition      % condition for when x1 < 0.5
            x2 = 2*parameter*x1;    % map operation, determine next iterate
            op = 0;                 % store symbol as 0
        elseif x1 > Map_partition   % condition for when x1 >= 0.5
            x2 = (2*parameter*x1)-1;% map operation, determine next iterate
            op = 1;                 % store symbol as 1
        end
        Real_Trajectories(i,n) = x1;      % stores the real ietrate
        Symbolic_Trajectories(i,n) = op;  % stores the symbol
        x1 = x2;                           % replaces old x1 with new x2

    end

    x = (x + increment);          % increases x by one step for next
                                    % initial condition
end

%---------- Map Plot ----------------------------------------------------
plot(Real_Trajectories(:,1),Real_Trajectories(:,2)); % plot map
set(gca,'xlim',[0 1]);                               % set axis views
set(gca,'ylim',[0 1]);
axis square;
```

Appendix 2.4: BM Bifurcation Diagram

Program for chaotic distribution of BM

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Bitshift Map bifurcation    ----------------------%%
%%-------------------- diagram                     ----------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Npre = 250;         % no. of initial few iterates thrown away for clear view
Nplot = 100;                % no. of points in an iteration to be plotted
x = zeros(Nplot,1);         % x trajectory array initialised
parameter = zeros(Nplot,1);   % parameter array initialised


for r = 0.0:0.00025:1.0       % for parameters sweeping [0,1] range
     x(1) = 0.5;              % set initial condition as 0.5
    for n = 1:Npre            % iterate the map for up to Npre
        if x(1) <= 0.5        % check x(1) is less than midpoint
            x(1) = 2*r*x(1);  % map operation, determine next iterat
                              % and overwrite
        elseif x(1) > 0.5     % check x(1) is greater than midpoint
            x(1) = (2*r*x(1))-1; % map operation, determine next iterat
                                 % and overwrite
        end

    end,
    for n = 1:Nplot-1               % iterate the map for up to Npre

        if x(n) <= 0.5              % map operation
```

```
            x(n+1) = 2*r*x(n);
        elseif x(n) > 0.5
            x(n+1) = (2*r*x(n))-1;
        end

    end
    plot(r*ones(Nplot,1), x, 'k.', 'markersize', 3); % ploting the iterates
    hold on;
end,

xlabel('μ');  ylabel('x_n');   % setting axis labels and viewing dimentions
set(gca, 'xlim', [0.75 1.05]);
set(gca, 'ylim', [-1 1]);

axis square;
hold off;
```

Appendix 2.5: Tent Map (TM)

Program for Tent Map (TM)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Tent Map generated for      ----------------------%%
%%-------------------- a set of initial conditions ----------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format longe

clear;                  % clears variables
clf;

Map_partition = 0.50;   % primary partition of the map
iteration = 10;          % setting number of iterations

Resolution = 8;         % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                          % input range [0,1] lower limit = 0
x_max = 1;                      % input range [0,1] upper limit = 1
xNew = x;                       % starting initial condition

N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions
                                  % within [0,1]

Real_Trajectories = zeros(N,iteration); % stores all iterates for N initial
                                        % conditions
Symbolic_Trajectories = zeros(N,iteration); % stores symbolic sequences for
                                              % N initialconditions
```

```
parameter = 1;                % sets the peak height of the map

for i = 1:N                   % runs for N number of initial conditions

    x1 = x;                   % copy initial condition to input variable

    for n = 1:iteration       % runs map for all iterations

        if x1 <= Map_partition      % condition for when x1 < 0.5
            x2 = 2*parameter*x1;    % map operation, determine next iterate
            op = 0;                 % store symbol as 0
        elseif x1 > Map_partition   % condition for when x1 >= 0.5
            x2 = 2*parameter*(1-x1);% map operation, determine next iterate
            op = 1;                 % store symbol as 1
        end
        Real_Trajectories(i,n) = x1;      % stores the real ietrate
        Symbolic_Trajectories(i,n) = op;  % stores the symbol
        x1 = x2;                           % replaces old x1 with new x2

    end

    x = (x + increment);          % increases x by one step for next
                                    % initial condition
end

%---------- Map Plot ----------------------------------------------------
plot(Real_Trajectories(:,1),Real_Trajectories(:,2)); % plot map
set(gca,'xlim',[0 1]);                               % set axis views
set(gca,'ylim',[0 1]);
axis square;
```

Appendix 2.6: TM Bifurcation Diagram

Program for TM distribution or bifurcation diagram

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Tent Map bifurcation        ----------------------%%
%%-------------------- diagram                     ----------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Npre = 250;         % no. of initial few iterates thrown away for clear view
Nplot = 100;                % no. of points in an iteration to be plotted
x = zeros(Nplot,1);         % x trajectory array initialised
parameter = zeros(Nplot,1);   % parameter array initialised


for r = 0.0:0.00025:1.0       % for parameters sweeping [0,1] range
     x(1) = 0.5;              % set initial condition as 0.5
    for n = 1:Npre            % iterate the map for up to Npre
        if x(1) <= 0.5        % check x(1) is less than midpoint
            x(1) = 2*r*x(1);  % map operation, determine next iterat
                              % and overwrite
        elseif x(1) > 0.5     % check x(1) is greater than midpoint
            x(1) = 2*r*(1-x(1)); % map operation, determine next iterat
                                 % and overwrite
        end

    end,
    for n = 1:Nplot-1               % iterate the map for up to Npre

        if x(n) <= 0.5              % map operation
```

```
            x(n+1) = 2*r*x(n);
        elseif x(n) > 0.5
            x(n+1) = 2*r*(1-x(n));
        end

% ------- noisy bifurcation diagram generated when uncommented ------------

%         x(n+1) = awgn(x(n+1),30);   % generates noisy dynamics
%         if x(n+1) < 0               % statespace limited within [0,1]
%             x(n+1) = 0.0001;
%         elseif x(n+1) > 1
%             x(n+1) = 0.999;
%         end

    end
    plot(r*ones(Nplot,1), x, 'k.', 'markersize', 3); % ploting the iterates
    hold on;
end,

xlabel('μ');  ylabel('x_n');   % setting axis labels and viewing dimentions
set(gca, 'xlim', [0.5 1]);
set(gca, 'ylim', [0 1]);

axis square;
hold off;
```

Appendix 2.7: TM Cobweb Diagrams

Program for TM cobweb diagrams

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Tent Map cobweb             ----------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format longe

clear;                  % clears variables
clf;

Map_partition = 0.50;   % primary partition of the map
iteration = 400;          % setting number of iterations

Resolution = 8;         % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                          % input range [0,1] lower limit = 0
x_max = 1;                      % input range [0,1] upper limit = 1
xNew = x;                       % starting initial condition

N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions
                                  % within [0,1]

Real_Trajectories = zeros(N,iteration); % stores all iterates for N initial
                                        % conditions
Symbolic_Trajectories = zeros(N,iteration); % stores symbolic sequences for
                                              % N initialconditions

parameter = 0.9;                % sets the peak height of the map
```

```
%------------------ framework of tentmap ---------------------------------

for i = 1:N                     % runs for N number of initial conditions

    x1 = x;                     % copy initial condition to input variable

    for n = 1:iteration         % runs map for all iterations

        if x1 <= Map_partition      % condition for when x1 < 0.5
            x2 = 2*parameter*x1;    % map operation, determine next iterate
            op = 0;                 % store symbol as 0
        elseif x1 > Map_partition   % condition for when x1 >= 0.5
            x2 = 2*parameter*(1-x1);% map operation, determine next iterate
            op = 1;                 % store symbol as 1
        end
        Real_Trajectories(i,n) = x1;      % stores the real ietrate
        Symbolic_Trajectories(i,n) = op;  % stores the symbol
        x1 = x2;                           % replaces old x1 with new x2

    end

    x = (x + increment);            % increases x by one step for next
                                      % initial condition
end

%---------------------- cobweb tent plot ---------------------------------

x1 = 0.157876 ;         % setting initial condition

if x1 <= Map_partition  % if initial condition is less than 0.5
   plot([x1,x1],[0,2*parameter*x1],'color',[0.0,0.5,0.8]); % plot next iter
else                                                       % if greater
   plot([x1,x1],[0,2*parameter*(1-x1)],'color',[0.0,0.5,0.8]); % plot iter
```

```
end
hold on;

for n = 1:iteration           % runs a for loop for the iterations

    if x1 <= Map_partition            % condition for when x < 0.5
        x2 = 2*parameter*x1;          % evaluates x for next iteration
        plot([x1,x1],[x2,x1],'color',[0.0,0.5,0.8]); % plot x1 to x2
        hold on;
        plot([x2,x1],[x2,x2],'color',[0.0,0.5,0.8]); % plot x2 on diagonal

    elseif x1 > Map_partition            % condition for when x >= 0.5
        x2 = 2*parameter*(1-x1);         % evaluates x for next iteration
        if(n<2)                          % generating cobweb
          plot([x1,x2],[x2,x2],'color',[0.0,0.5,0.8]);
        else
          plot([x1,x1],[x2,x1],'color',[0.0,0.5,0.8]);  % plot x1 to x2
        end
        hold on;
        plot([x2,x1],[x2,x2],'color',[0.0,0.5,0.8]);  % plot x2 on diagonal

    end
    x1 = x2;   % feedback iterates
end

%------------- plot tent map frame ----------------------------------------
plot(Real_Trajectories(:,1),Real_Trajectories(:,1),'k');
plot(Real_Trajectories(:,1),Real_Trajectories(:,2),'k');

%------------- axes configuration -----------------------------------------
set(gca,'xlim',[0 1]);
set(gca,'ylim',[0 1]);
axis square;
```

Appendix 2.8: GON for TM

Program for Gray Ordering Number (GON) of symbolic sequence generated by TM

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%--------------------- GON Estimation (Tent Map)   ----------------------%%
%%--------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format longe

clear;                   % clears variables
clf;

Map_partition = 0.50;    % primary partition of the map
iteration = 30;           % setting number of iterations

Resolution = 8;          % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                           % input range [0,1] lower limit = 0
x_max = 1;                       % input range [0,1] upper limit = 1
xNew = x;                        % starting initial condition

N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions
                                  % within [0,1]

Real_Trajectories = zeros(N,iteration); % stores all iterates for N initial
                                        % conditions
Gray_Traj = zeros(N,iteration); % stores symbolic sequences for
                                          % N initialconditions
Binary = zeros(N,iteration);    % stores all binary sequence
GON = zeros(N,1);               % stores all binary GON
```

```
parameter = 0.8;                % sets the peak height of the map

for i = 1:N                     % runs for N number of initial conditions

    x1 = x;                     % copy initial condition to input variable

    for n = 1:iteration         % runs map for all iterations

        if x1 <= Map_partition      % condition for when x1 < 0.5
            x2 = 2*parameter*x1;    % map operation, determine next iterate
            op = 0;                 % store symbol as 0
        elseif x1 > Map_partition   % condition for when x1 >= 0.5
            x2 = 2*parameter*(1-x1);% map operation, determine next iterate
            op = 1;                 % store symbol as 1
        end
        Real_Trajectories(i,n) = x1;      % stores the real ietrate
        Gray_Traj(i,n) = op;  % stores the symbol
        x1 = x2;                          % replaces old x1 with new x2

    end

    x = (x + increment);            % increases x by one step for next
                                      % initial condition
end

for row = 1:N                   % for all initial conditions
    for col = 1:iteration       % for all iterates
        if col == 1             % converting gray to binary
            Binary(row,col) = Gray_Traj(row,col);
        elseif col > 1
            Binary(row,col) = bitxor(Gray_Traj(row,col),Binary(row,col-1));
        end
    end
```

```
    for col = 1:iteration       % estimating GON
        GON(row,1) = (GON(row,1)+(Binary(row,col)*(2^(-(col)))));
    end
end
GON(:,2) = (Real_Trajectories(:,1) - GON(:,1))*100; % error in GON estimate
hold on;
plot(Real_Trajectories(:,1),GON(:,1),'k');      % plotting GON
plot(Real_Trajectories(:,1),Real_Trajectories(:,1),'k--');
                                                % plotting initial condition
axis square;
```

Appendix 2.9: TM Shifting Window

Program for shifting window over TM generated symbolic sequence

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%--------------------- Shifting window (Tent Map)  ----------------------%%
%%--------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format longe

clear;                    % clears variables
clf;

Map_partition = 0.50;     % primary partition of the map
iteration = 30;            % setting number of iterations

Resolution = 8;           % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                          % input range [0,1] lower limit = 0
x_max = 1;                      % input range [0,1] upper limit = 1
xNew = x;                       % starting initial condition

N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions
                                  % within [0,1]

Real_Trajectories = zeros(N,iteration); % stores all iterates for N initial
                                        % conditions
Gray_Traj = zeros(N,iteration); % stores symbolic sequences for
                                          % N initialconditions
Binary = zeros(N,iteration);    % stores all binary sequence
GON = zeros(N,1);               % stores all binary GON
```

```
parameter = 0.8;                % sets the peak height of the map

% ------------- TM Trajectory generation ---------------------------------

for i = 1:N                     % runs for N number of initial conditions

    x1 = x;                     % copy initial condition to input variable

    for n = 1:iteration         % runs map for all iterations

        if x1 <= Map_partition      % condition for when x1 < 0.5
            x2 = 2*parameter*x1;    % map operation, determine next iterate
            op = 0;                 % store symbol as 0
        elseif x1 > Map_partition   % condition for when x1 >= 0.5
            x2 = 2*parameter*(1-x1);% map operation, determine next iterate
            op = 1;                 % store symbol as 1
        end
        Real_Trajectories(i,n) = x1;      % stores the real ietrate
        Gray_Traj(i,n) = op;  % stores the symbol
        x1 = x2;                           % replaces old x1 with new x2

    end

    x = (x + increment);            % increases x by one step for next
                                      % initial condition
end

%----------------- Symbolic dynamics processing ---------------------------
window_size = 8;                  % declare window size
Win_GON = zeros(N,iteration);     % Window GON array initialised
Win_Bin = zeros(1,window_size);   % Window binary array initialised

for row = 1:N                   % for all initial conditions
```

```
    for col = 1:iteration       % for all iterates
        if col == 1             % converting gray to binary
            Binary(row,col) = Gray_Traj(row,col);
        elseif col > 1
            Binary(row,col) = bitxor(Gray_Traj(row,col),Binary(row,col-1));
        end
    end

    for col = 1:iteration       % estimating GON
        GON(row,1) = (GON(row,1)+(Binary(row,col)*(2^(-(col)))));
    end

% ------------ shifting window ------------------------------------------
    for col = 1:iteration  % shifting 1 place for all symbols in a sequence
        for no = col:col+(window_size-1)   % for all symbols within window
            if col<=iteration-(window_size-1) % check if it isnt last shift
                if ((no - col) == 0)   % converting window symbol to binary
                    Win_Bin(1,(no-col)+1) = Gray_Traj(row,no);
                elseif ((no - col) > 0)
                    Win_Bin(1,(no-col)+1) = bitxor(Gray_Traj(row,no),Win_Bin(1,no-col));
                end
            end
        end
        for num = 1:window_size    % calculating GON for the window symbol
            if col<=iteration-(window_size-1)
                Win_GON(row,col) = Win_GON(row,col)+(Win_Bin(1,num)*(2^(-(num))));
            end
        end
    end
end
GON(:,2)= (Real_Trajectories(:,1) - GON(:,1))*100; % difference between GON
                                                   % and actual

%----------Plot generation-----------------------------------------------
hold on
```

```
% plot(Real_Trajectories(:,1),GON(:,1),'k');
% plot(Real_Trajectories(:,1),Real_Trajectories(:,1),'k--');
plot(Real_Trajectories(152,:),'b','Markersize',2);
plot(Win_GON(152,:),'r','Markersize',2);
```

Appendix 2.10: Kneading Sequence Method (single input)

Program for parameter estimation through Kneading sequence search algorithm (for single initial condition)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Parameter Estimation Algorithm -------------------%%
%%-------------------- Using shifting window          -------------------%%
%%-------------------- Program operated for a single input --------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long

%///////////////// Symbolic Data Generated Using Non-ideal TM /////////////

iteration = 200;    % number of iterations initialised
partition = 0.5;    % map partition defined

x = 0.092904;       % initial condition chosen for the experiment
parameter = 0.8;    % parameter chosen for the experiment

Real_Trajectories = zeros(1,iteration); % initialise real trajectory array
Gray_Traj = zeros(1,iteration);    % initialise symbolic trajectory array

for n = 1:iteration                % runs map for all iterations
    if x <= partition              % condition for when x1 < 0.5
        x2 = 2*parameter*x;        % map operation, determine next iterate
        sym = 0;                   % store symbol as 0
    elseif x > partition           % condition for when x >= 0.5
        x2 = 2*parameter*(1-x);    % map operation, determine next iterate
        sym = 1;                   % store symbol as 1
    end
```

```
    Real_Trajectories(1,n) = x;  % store the x for iteration
    Gray_Traj(1,n) = sym;      % store the op for iteration
    x = x2;                     % replaces old x with new x
end

%////////////// Kneading Sequence Search Through Shifting Window /////////////

window_size = 12;                          % declare window size
Transient_beta = 5;                        % declare transient beta symbols
Win_Bin = zeros(1,window_size);            % initialise window binary array
Win_Gray = zeros(1,window_size);           % initialise window gray array
Smax = zeros(1,window_size);               % initialise Smax register
Kneading_sequence = zeros(1,window_size+1); % initialise kneading sequence
                                             % register
GON_window = 0;                       % initialise GON for window shifts
Largest = 0;                          % initialise largest tracking variable

for col = Transient_beta + 1:iteration % for each symbol after discarding
                                       % dscarding beta symbols
  for no = col:col+(window_size-1) % for symbols within the shifted window
      if col<=iteration-(window_size-1) % check if it isn't the last shift
        if ((no - col) == 0)  % convert window symbol to binary
         Win_Bin(1,(no-col)+1) = Gray_Traj(1,no);
        elseif ((no - col) > 0)
         Win_Bin(1,(no-col)+1) = bitxor(Gray_Traj(1,no),Win_Bin(1,no-col));
        end
        Win_Gray(1,(no-col)+1)= Gray_Traj(1,no); % alo save window gray
      end
  end
  for num = 1:window_size          % calculate GON for the window sequence
      if col<=iteration-(window_size-1)
        GON_window = GON_window +(Win_Bin(1,num)*(2^((window_size-num))));
      end
  end
```

```
  if GON_window > Largest    % track the largest GON by comparing previous
                             % largest to current GON
     Largest = GON_window;   % update largest if greater GONs found
     Smax = Win_Gray;        % store window gray as Smax for largest GON
     Iterate_location = col;     % point at which iteration the largest was
                                 % found
  end
  GON_window = 0;            % reset GON of window for the next shift
end

%---------------- Preparing the Kneading Sequence -------------------------
Kneading_sequence(1,2:end) = Smax; % storing Smax from the 2nd position so
                                   % a 0 automatically added in the 1st
                                     % place
K_length = window_size+1; % size of kneading sequence udated after adding 0


%/////////// Solving the polynomial equation with GON and x_c = 0.5 /////////

Count_one = 0;            % variable to count odd even 1s initialised
Bin = zeros(1,K_length); % binary register of kneading sequnce initialised
GON = 0;                  % GON of kneading sequence
Equation = zeros(1,K_length);  % difference equation array initialised
Estimated_mu = 0;         % estimated parameter variable initialied

%------------Finding GON of Kneading sequence -----------------------------
    for col = 1:K_length
        if col == 1           % gray to binary estimation
            Bin(1,col) = Kneading_sequence(1,col);
        elseif col > 1
            Bin(1,col) = bitxor(Kneading_sequence(1,col),Bin(1,col-1));
        end
        GON = (GON +(Bin(1,col)*(2^(-(col))))); % calculate GON
```

```
    end

%---------- Forming the difference equation with signs of delta -----------

for col = 1:K_length                                    % for all symbols in K
    Count_one = Count_one + Kneading_sequence(1,col);% count number of ones
    if rem(Count_one,2) == 0                            % check odd/even
      Equation(col) = (-1)*Kneading_sequence(col);    % negative when even
    else
      Equation(col) = (1)*Kneading_sequence(col);     % positive when odd
    end
end

  constant = 0.5 - GON;     % determine the cnstant part of the polynomial
  for col = 1:K_length
      constant = constant + (Equation(col)*2^(-(col-1))); % further update
                            % the constant part with 2^i for all the
                            % differences
  end

  Equation(1) = -constant;  % store the constant in the equation array
                              % with remaining order of polynomial
                                % coefficients intact
  Root = roots(Equation);   % solving the equation

for r = 1:K_length-1  % check within the number (K_length-1) of roots
    if (real(Root(r,1)) > 0) && (imag(Root(r,1))==0 && (real(Root(r,1))> Estimated_mu*2))
                                % select largest non-complex root
        Estimated_mu = Root(r,1)/2;  % derive the parameter from 2mu part
    end
end
```

Appendix 2.11: Kneading Sequence Method (full dataset)

Program for parameter estimation through Kneading sequence search algorithm (for all initial condition in a dataset)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Parameter Estimation Algorithm -------------------%%
%%-------------------- Using shifting window          -------------------%%
%%-------------------- Program operated for all input -------------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long

%///////////////// Symbolic Data Generated Using Non-ideal TM /////////////

iteration = 200;   % number of iterations initialised
partition = 0.5;   % map partition defined

Resolution = 8;         % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                          % input range [0,1] lower limit = 0
x_max = 1;                      % input range [0,1] upper limit = 1

N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions
                                  % within [0,1]


parameter = 0.6;   % parameter chosen for the experiment

Real_Trajectories = zeros(N,iteration); % initialise real trajectory array
Gray_Traj = zeros(N,iteration);   % initialise symbolic trajectory array
```

```
for row = 1:N                          % for all initial conditions
    x1 = x;                            % start with an intial condition
    for n = 1:iteration                % runs map for all iterations
        if x1 <= partition              % condition for when x1 < 0.5
            x2 = 2*parameter*x1;      % map operation, determine next iterate
            sym = 0;                   % store symbol as 0
        elseif x1 > partition           % condition for when x >= 0.5
            x2 = 2*parameter*(1-x1);% map operation, determine next iterate
            sym = 1;                   % store symbol as 1
        end
        Real_Trajectories(row,n) = x1;  % store the x for iteration
        Gray_Traj(row,n) = sym;        % store the sym for iteration
        x1 = x2;                       % replaces old x with new x
    end
       x = (x + increment);             % increases x by one step for next
                                       % initial condition
end

%////////////// Kneading Sequence Search Through Shifting Window //////////////

window_size = 12;                           % declare window size
Transient_beta = 5;                         % declare transient beta symbols
Win_Bin = zeros(1,window_size);             % initialise window binary array
Win_Gray = zeros(1,window_size);            % initialise window gray array
Smax = zeros(N,window_size);                % initialise Smax register
Kneading_sequence = zeros(N,window_size+1); % initialise kneading sequence
                                              % register
GON_window = 0;                        % initialise GON for window shifts
Largest = 0;                           % initialise largest tracking variable

for row = 1:N                           % for all initial conditions
    for col = (Transient_beta + 1):iteration % for each symbol after
                                           % dscarding beta symbols
        for no = col:col+(window_size-1) % for symbols within the window
```

```
            if col<=iteration-(window_size-1) % check if isn't last shift
                if ((no - col) == 0)  % convert window symbol to binary
                    Win_Bin(1,(no-col)+1) = Gray_Traj(row,no);
                elseif ((no - col) > 0)
                    Win_Bin(1,(no-col)+1) = bitxor(Gray_Traj(row,no),Win_Bin(1,no-col));
                end
                Win_Gray(1,(no-col)+1)= Gray_Traj(row,no);
                                                % alo save window gray
            end
        end
        for num = 1:window_size     % calculate GON for the window sequence
            if col<=iteration-(window_size-1)
                GON_window = GON_window +(Win_Bin(1,num)*(2^((window_size-num))));
            end
        end

        if GON_window > Largest   % track the largest GON by comparing
                                  % previous largest to current GON
            Largest = GON_window; % update largest if greater GONs found
            Smax(row,:) = Win_Gray(1,:); % store window gray as Smax for
                                         % largest GON
        end
        GON_window = 0;           % reset GON of window for the next shift
    end
Largest = 0;
%---------------- Preparing the Kneading Sequence ------------------------
Kneading_sequence(row,2:end) = Smax(row,:);
                                  % storing Smax from the 2nd position so
                                     % a 0 automatically added in the 1st
                                       % place
end

%/////////// Solving the polynomial equation with GON and x_c = 0.5 /////////

K_length = window_size+1; % size of kneading sequence udated after adding 0
```

```
Count_one = 0;           % variable to count odd even 1s initialised
Bin = zeros(N,K_length); % binary register of kneading sequnce initialised
GON = 0;                 % GON of kneading sequence
Equation = zeros(1,K_length);  % difference equation array initialised
Estimated_mu = zeros(N,1);        % estimated parameter variable initialied

for row = 1:N                % for all input
    GON = 0;                 % GON of kneading sequence
    Equation = zeros(1,K_length);  % difference equation array initialised
    Count_one = 0;           % variable to count odd even 1s initialised
    %------------Finding GON of Kneading sequence -------------------------
    for col = 1:K_length     % for all symbols in K
        if col == 1          % gray to binary estimation
            Bin(1,col) = Kneading_sequence(row,col);
        elseif col > 1
            Bin(1,col) = bitxor(Kneading_sequence(row,col),Bin(1,col-1));
        end
        GON = (GON +(Bin(1,col)*(2^(-(col))))); % calculate GON
    end

    %---------- Forming the difference equation with signs of delta -------

    for col = 1:K_length                              % for all symbols in K
        Count_one = Count_one + Kneading_sequence(row,col);% no. of 1s
        if rem(Count_one,2) == 0                          % check odd/even
            Equation(col) = (-1)*Kneading_sequence(row,col);% -ive if even
        else
            Equation(col) = (1)*Kneading_sequence(row,col); % +ive if odd
        end
    end

    constant = 0.5 - GON;   % determine the cnstant part of the polynomial
    for col = 1:K_length
        constant = constant + (Equation(col)*2^(-(col-1)));
        % further update the constant part with 2^i for all the differences
```

```
    end

    Equation(1) = -constant;  % store the constant in the equation array
                              % with remaining order of polynomial
                                % coefficients intact
    Root = roots(Equation);   % solving the equation

    for r = 1:K_length-1  % check within the number (K_length-1) of roots
        if (real(Root(r,1)) > 0) && (imag(Root(r,1))==0 && (real(Root(r,1))> Estimated_mu(row,1)*2))
            % select largest non-complex root
            Estimated_mu(row,1) = Root(r,1)/2;  % derive the parameter from 2mu part
        end
    end
end
% plot(Real_Trajectories(2:256,1),Estimated_mu(2:256,1));
% set(gca,'ylim',[0.7 0.9]);
axis square
hold on
plot(Real_Trajectories(:,1),Real_Trajectories(:,2));
```

Appendix 2.12: Crossover Method (single input)

Program for parameter estimation through crossover detection within noisy field (for single initial condition)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Parameter Estimation Algorithm -------------------%%
%%-------------------- from crossovers in noisy trajectories ------------%%
%%-------------------- Program operated for single input ----------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long

%////////////// TM Noisy Dataset Generation For Test conditions ////////////

%------------------ Tunable variables -------------------------------------
iteration = 50;             % no of iterates in a trajectory
samples = 50;               % no of sampled observations per trajectory
x = 0.86328125;         % chosen initial condition
parameter = 0.90;     % chosen map paramter
SNR_db = 30;             % noise level in every stage of iteration
%------------------ END of Tunable variables ------------------------------
% initialisations
partition = 0.5;                                % map partition
eta = zeros(samples,iteration); % iterative trajectories
Gray_Traj = zeros(samples,iteration);         % symbolic trajectories

%----------------- Map operation with noise -------------------------------

for i = 1:samples                   % for a given sample
    x1 = x;                         % start with initial condition
    for n = 1:iteration             % runs map for all iterations
```

```
        x1 = awgn(x1,SNR_db);
        if x1 <= partition          % condition for when x1 < 0.5
            x2 = 2*parameter*x1;    % map operation, determine next iterate
            Sym = 0;                % store symbol as 0
        elseif x1 > partition       % condition for when x >= 0.5
            x2 = 2*parameter*(1-x1);% map operation, determine next iterate
            Sym = 1;                % store symbol as 1
        end
        eta(i,n) = x1;  % store the x for iteration
        Gray_Traj(i,n) = Sym;         % store the sym for iteration

        x1 = x2;            % replacing old x with new x, map feedback
        if x1 < 0           % clamp noisy field between [0,1] in statespace
            x1 = 0.0001;
        elseif x1 > 1
            x1 = 0.999;
        end

    end
end

%---------------------- Plot noisy data ----------------------------------
clf
hold on
for samp = 1:samples
    plot(eta(samp,1:iteration),'color',[0.5 0.5 0.5],'Markersize',1);
end

%///////////////// Crossover Analysis & Parameter Estimation ///////////////

index = 1;      % initialise index counter for sorting routines
col_count = 1; % initialise column count variable
sol_xy = zeros((samples*(samples-1)/2),2); % XY solution array for each m
Fix_chase_x = zeros(1,iteration); % estimated intersection over x axis
Fix_chase_y = zeros(1,iteration); % estimated intersection over y axis
```

```
sol_count_x = 0;                  % initialised count of no. of solutions

for n = 1:(iteration - 1) % for all the iterations

%------- solving straightline equations to determine intersection ---------

    for m = 1:samples - 1   % considering one sample at a time
        for t = m + 1:samples % considering other samples
            if(eta(m,n)<=1 && eta(m,n)>=0.5 && eta(t,n)<=1 && eta(t,n)>=0.5) % check if samples comply 0.5<=eta<=1
                                                                                                    for Hn set
                sol_xy(index,1) = ((eta(m,n) - eta(t,n))/(eta(m,n) - eta(m,n+1) - eta(t,n) + eta(t,n+1))) + n;
                                                                                % solve for x coordinate
                sol_xy(index,2) = ((eta(t,n)*(eta(m,n) - eta(m,n+1))) - (eta(m,n)*(eta(t,n) -
eta(t,n+1))))/(eta(m,n) - eta(m,n+1) - eta(t,n) + eta(t,n+1));
                                        % solve for y coordinates between straight lines formed by n and n+1 samples
                index = index + 1; % counting number of solutions
            end
        end
    end
    sol_xy = sortrows(sol_xy(1:index - 1,:)); % sort x solution array to separate out no solutions or 0s
    Mean_X = nanmean(sol_xy(1:index - 1,1));  % take average of x solution for nth step excluding NaNs
    X_SD = std(sol_xy(1:index - 1,1));        % determine standard deviation of x solutions
    edgex_L = Mean_X - X_SD;       % determine lower bound for x solutions
    edgex_H = Mean_X + X_SD;       % determine upper bound for x solutions

    sol_count_x = index - 1;  % store count of x solutions
    select_y = zeros(index,1); % initialise array for selected y solutions
    index = 1;          % initialise index for sorting y solutions

    for row = 1:sol_count_x  % for the number of non-zero x solutions
        if sol_xy(row,1) >= edgex_L && sol_xy(row,1) <= edgex_H % checking if x solution belongs in the boundary
            select_y(index,1) = sol_xy(row,2); % select corresponding y solution (for all meaningful x solutions)
            index = index + 1;    % count number of y solutions
        end
    end
```

```
    Mean_Y = nanmean(select_y(1:index - 1,1)); % mean of y solutions to determine a central point of intersections
    Y_SD = std(select_y(1:index - 1,1)); % determine standard deviation of y solutions

    Fix_chase_x(1,col_count) = n;     % fill array for nth location of the solution
    Fix_chase_y(1,col_count) = Mean_Y; % mean of all y solutions at nth location (ybar in algorithm)
    col_count = col_count + 1;         % count number of meaningful solutions
    Chase = sort(Fix_chase_y(1,1:col_count-1)); % sort all the solutions to separate out 0
    gain = nanmean(Chase)/(1-nanmean(Chase)); % determine the 2mu part
    index = 1;                                % reset variables for next nth solution
    sol_xy = zeros((samples*(samples-1)/2),2);
    sol_count_x = 0;
end

Mean_Chase(1:iteration) = mean(Chase); % determine the mean of all crossover to locate fixed pont
SD_Chase = std(Chase);
Estim_para = gain/2;                   % estimating the parameter

% ---------plot the mean chase (fixed point estimate from crossover) ------
hold on
plot(1:iteration,Mean_Chase(1:iteration),'k');
plot(Fix_chase_x(1,1:iteration-1),Fix_chase_y(1,1:iteration-1),'ks--','Markersize',7,'markerfacecolor',[0,0,0]);
axis square
```

Appendix 2.13: Crossover Method (full dataset)

Program for parameter estimation through crossover detection within noisy field (for all initial condition)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%-------------------- Parameter Estimation Algorithm -------------------%%
%%-------------------- from crossovers in noisy trajectories ------------%%
%%-------------------- Program operated for all input    ----------------%%
%%-------------------- Author: Dhrubajyoti Dutta   ----------------------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long

%////////////// TM Noisy Dataset Generation For Test conditions ////////////

%------------------ Tunable variables -------------------------------------
iteration = 50;             % no of iterates in a trajectory
samples = 50;               % no of sampled observations per trajectory
parameter = 0.85;     % chosen map paramter
SNR_db = 30;              % noise level in every stage of iteration
%------------------ END of Tunable variables ------------------------------
% initialisations
partition = 0.5;                                % map partition
Resolution = 8;         % resolution of initial data set
increment = (1/(2^Resolution)); % size of initial data as 1/2^Resolution
x = 0;                          % input range [0,1] lower limit = 0
x_max = 1;                      % input range [0,1] upper limit = 1
N = ceil((x_max-x)/increment)+1;  % calculates number of initial conditions within [0,1]
eta = zeros(samples,iteration,N); % iterative trajectories
Gray_Traj = zeros(samples,iteration,N);         % symbolic trajectories

%---------------- Map operation with noise ------------------------------
```

```
for k = 1:N
    for i = 1:samples                   % for a given sample
        x1 = x;                         % start with initial condition
        for n = 1:iteration             % runs map for all iterations
            x1 = awgn(x1,SNR_db);
            if x1 <= partition          % condition for when x1 < 0.5
                x2 = 2*parameter*x1;    % map operation, determine next iterate
                Sym = 0;                % store symbol as 0
            elseif x1 > partition       % condition for when x >= 0.5
                x2 = 2*parameter*(1-x1);% map operation, determine next iterate
                Sym = 1;                % store symbol as 1
            end
            eta(i,n,k) = x1;  % store the x for iteration
            Gray_Traj(i,n,k) = Sym;         % store the sym for iteration
                        x1 = x2;           % replacing old x with new x, map feedback
            if x1 < 0          % clamp noisy field between [0,1] in statespace
                x1 = 0.0001;
            elseif x1 > 1
                x1 = 0.999;
            end
        end
    end
    x = x + increment;
end

%///////////////// Crossover Analysis & Parameter Estimation ///////////////

index = 1;     % initialise index counter for sorting routines
col_count = 1; % initialise column count variable
sol_xy = zeros((samples*(samples-1)/2),2); % XY solution array for each m
Fix_chase_x = zeros(1,iteration); % estimated intersection over x axis
Fix_chase_y = zeros(1,iteration); % estimated intersection over y axis
sol_count_x = 0;                  % initialised count of no. of solutions
Estim_Parameter = zeros(N,1);
gain = 0;
```

```
for k = 1:N                    % for all input conditions
    for n = 1:(iteration - 1) % for all the iterations

%------- solving straightline equations to determine intersection ---------

        for m = 1:samples - 1   % considering one sample at a time
            for t = m + 1:samples % considering other samples
                if(eta(m,n,k)<=1 && eta(m,n,k)>=0.5 && eta(t,n,k)<=1 && eta(t,n,k)>=0.5) % check if samples comply
                                                                                    0.5<=eta<=1 for Hn set
                    sol_xy(index,1) = ((eta(m,n,k) - eta(t,n,k))/(eta(m,n,k) - eta(m,n+1,k) - eta(t,n,k) +
eta(t,n+1,k))) + n; % solve for x coordinate
                    sol_xy(index,2) = ((eta(t,n,k)*(eta(m,n,k) - eta(m,n+1,k))) - (eta(m,n,k)*(eta(t,n,k) -
eta(t,n+1,k))))/(eta(m,n,k) - eta(m,n+1,k) - eta(t,n,k) + eta(t,n+1,k));
                    % solve for y coordinates between straight lines formed by n and n+1 samples
                    index = index + 1; % counting number of solutions
                end
            end
        end
        sol_xy = sortrows(sol_xy(1:index - 1,:)); % sort x solution array to separate out no solutions or 0s
        Mean_X = nanmean(sol_xy(1:index - 1,1));  % take average of x solution for nth step excluding NaNs
        X_SD = std(sol_xy(1:index - 1,1));        % determine standard deviation of x solutions
        edgex_L = Mean_X - X_SD;       % determine lower bound for x solutions
        edgex_H = Mean_X + X_SD;       % determine upper bound for x solutions
        sol_count_x = index - 1;  % store count of x solutions
        select_y = zeros(index,1); % initialise array for selected y solutions
        index = 1;          % initialise index for sorting y solutions

        for row = 1:sol_count_x  % for the number of non-zero x solutions
            if sol_xy(row,1) >= edgex_L && sol_xy(row,1) <= edgex_H % checking if x solution belongs in the boundary
                select_y(index,1) = sol_xy(row,2); % select corresponding y solution (for all meaningful x
solutions)
                index = index + 1;    % count number of y solutions
            end
        end
```

```
        Mean_Y = nanmean(select_y(1:index - 1,1)); % mean of y solutions to determine a central point of
intersections
        Y_SD = std(select_y(1:index - 1,1)); % determine standard deviation of y solutions

        Fix_chase_x(1,col_count) = n;     % fill array for nth location of the solution
        Fix_chase_y(1,col_count) = Mean_Y; % mean of all y solutions at nth location (ybar in algorithm)
        col_count = col_count + 1;         % count number of meaningful solutions
        Chase = sort(Fix_chase_y(1,1:col_count-1)); % sort all the solutions to separate out 0
        gain = nanmean(Chase)/(1-nanmean(Chase)); % determine the 2mu part
        index = 1;                                % reset variables for next nth solution
        sol_xy = zeros((samples*(samples-1)/2),2);
        sol_count_x = 0;
    end
    Mean_Chase(1:iteration) = mean(Chase); % determine the mean of all crossover to locate fixed pont
    SD_Chase = std(Chase);
    Estim_para = gain/2;                   % estimating the parameter
    Estim_Parameter(k,1) = Estim_para; % estimated parameter for all initial conditions

    %----- reset all variables for the estimation of the next input -------
    index = 1;
    col_count = 1;
    sol_xy(:,:) = 0;
    Fix_chase_x(:,:) = 0;
    Fix_chase_y(:,:) = 0;
    Chase(:,:) = 0;
    sol_count_x = 0;
    clear select_y;
    Mean_X = 0;
    X_SD = 0;
    edgex_L = 0;
    edgex_H = 0;
    Mean_Y = 0;
    Y_SD = 0;
end
```